

普通高等教育“十一五”国家级规划教材



21世纪大学本科计算机专业系列教材

宋佳兴 王诚 编著

计算机组成与体系结构（第3版）

——基本原理、设计技术与工程实现

http://www.tup.com.cn

- 根据教育部“高等学校计算机科学与技术专业规范”组织编写
- 与美国 ACM 和 IEEE CS *Computing Curricula* 最新进展同步
- 国家精品课程教材

清华大学出版社

普通高等教育“十一五”国家级规划教材
21世纪大学本科计算机专业系列教材

计算机组成与体系结构（第3版）

——基本原理、设计技术与工程实现

宋佳兴　王　诚　编著

清华大学出版社
北　京

内 容 简 介

本书包括数字电路基础(先修部分)、计算机组成(主体部分)、计算机体系结构(提高部分)3部分内容,共13章,重点讲解计算机系统的完整组成和提高性能的可行途径。作为硬件课程教材,兼顾到计算机科学与技术专业中偏工程技术方向、偏软件方向的本科生,也可用于软件学院和计算机应用方向的学生。

图书在版编目(CIP)数据

计算机组成与体系结构:基本原理、设计技术与工程实现/宋佳兴,王诚编著. —3版. —北京:清华大学出版社,2017
(21世纪大学本科计算机专业系列教材)
ISBN 978-7-302-46554-6

Ⅰ. ①计…　Ⅱ. ①宋…　②王…　Ⅲ. ①计算机体系结构-高等学校-教材　Ⅳ. ①TP303

中国版本图书馆CIP数据核字(2017)第030105号

责任编辑:张瑞庆　薛　阳
封面设计:常雪影
责任校对:梁　毅
责任印制:刘海龙

出版发行:清华大学出版社
网　　址:http://www.tup.com.cn, http://www.wqbook.com
地　　址:北京清华大学学研大厦A座　　**邮　编**:100084
社 总 机:010-62770175　　**邮　购**:010-62786544
投稿与读者服务:010-62776969,c-service@tup.tsinghua.edu.cn
质 量 反 馈:010-62772015,zhiliang@tup.tsinghua.edu.cn
课 件 下 载:http://www.tup.com.cn,010-62795954
印 装 者:北京泽宇印刷有限公司
经　　销:全国新华书店
开　　本:185mm×260mm　　**印　张**:20.75　　**字　数**:504千字
版　　次:2004年1月第1版　2017年6月第3版　　**印　次**:2017年7月第1次印刷
印　　数:1～2000
定　　价:49.00元

产品编号:071041-01

前　言

FOREWORD

对这一版的书名做了一点变动，增加了“基本原理、设计技术与工程实现”的副标题，特意指出书中提供3类不同性质的教学内容。第一类是计算机组成原理与运行机制的核心知识，属于学生一定要掌握的原理性基础知识，在绝大多数的同类教材中都会重点讲解。第二类是计算机硬件系统的基本设计技术，多数同类教材中讲解不多也欠具体应用实例。第三类是计算机硬件系统的入门性工程实现问题，同类教材中较少涉及，我们希望在这一版的教材中对上述3类内容都有适度的讲解，其中的工程实现问题只在主教材中简单提及，主要部分将放到《计算机组成与体系结构实验指导》教材中。这种安排体现了作者多年坚守的教学理念，从如下3个方面予以说明。

（1）针对计算机组成原理这一类含有较多技术性、工程性、实践性内容的课程，教学安排不宜过分局限于教师课堂讲课、学生课上听讲和课后背书的学习方式，应该在讲课听课的基础上再较大幅度地加强教学实践环节，增强课程内容的实用性，促成学生用课堂学到的理论知识，设计实现一台组成简单完整、原理清晰实用、实验操作方便、支持汇编语言编程的简小计算机系统，促使学生在学习理论知识、掌握设计技术、提升实践能力等诸方面得到全面成长。

（2）在讲解计算机硬件系统组成和功能实现时，应该把口语性的一般讲解和硬件描述语言的严谨描述恰当地结合起来，鼓励选用硬件描述语言设计计算机控制器和描述整机系统，既能体现计算机的最新设计技术和实现手段，保证教学内容适度的先进性，又可以培养学生严谨的思维方式和对硬件问题的深入理解和准确阐述，这有利于提高授课质量，降低学习和实验的难度。

（3）在课程的教学安排中，需要处理好硬件子系统（重点部分）和软件子系统（配合部分）的关系，不能完全局限于硬件系统本身，还需要包含必要的软件内容，汇编语言程序设计应该占有一定分量，加深对硬、软件两类资源各自在计算机系统中的地位和作用的理解。指令系统是连接硬、软件系统的纽带，汇编语言编程有助于深入了解指令系统、计算机整机组成与运行控制机制。因此在教学计算机系统中配备了3个基本程序：PC仿真终端程序、交叉汇编程序、监控程序，对多数同学来说做到会使用它们就够了，鼓励有余力的同学探索这几个程序的实现思路和方法。

本书包括了数字电路基础、计算机组成、计算机体系结构3部分内容，共13章。

第1章是全书内容的概述部分，简要介绍计算机组成和体系结构的基本概念，从实现功能的角度介绍计算机硬件系统的5个功能部件；从功能和层次的观点来讲解计算机组成和

体系结构各自需要研究和解决的问题,并简要说明了本课程的教学目标和对学习方法的建议。

第2章简明讲解数字电路基础知识和几种常用的电路芯片,是为讲解计算机组成和体系结构做电子线路方面的准备,没有这些知识是很难学懂计算机硬件的组成和运行原理的。

第3章的数据表示和运算、第4章的运算器部件共同构成本书核心内容的第1个知识单元,主要围绕承担数据运算功能的运算器部件进行讲解,在给出通用的基本原理知识的同时,还提供了设计实现一个原理性的8位运算器模型和一个4位位片结构的运算器芯片两个实例,展现运算器部件的设计过程和实现方法,提升学生的实践能力。

第5章的指令系统和第6章的控制器部件共同构成本书核心内容的第2个知识单元,主要围绕指令格式选择、指令系统设计,以及硬件系统中的硬布线方案的控制器部件进行讲解,而对微程序方案控制器只作适当介绍。在给出通用的基本原理知识的同时,提供了一套简单实用的基本指令系统。硬件方面,选用多指令周期方案实现这套指令系统的控制器部件的具体例子,展现控制器部件的设计过程和实现方法,提升学生的实践能力。软件方面,使用这套指令系统设计了教学机的监控程序,可以支持汇编语言程序设计;若再扩展一部分指令,也可以支持解释执行的BASIC高级语言程序设计,能支持浮点数运算和多种基本数学函数运算。针对这门课程的教学要求来说,此时的教学机的硬软件系统已经比较完整,包括了计算机硬件软件系统全部6个层次的基本内容。

第7章的主存、第8章的高速缓存和虚存、第9章的辅助存储器设备共同构成本书核心内容的第3个知识单元,主要围绕计算机3级结构的存储器件系统和外存储器设备进行讲解,还给出了通过字、位扩展技术,用静态芯片构建内存储器部件的具体例子,支持存储器与CPU同步运行,展现内存储器的功能和经总线连接CPU的具体方法。

第10章的输入输出设备和第11章的输入输出系统共同构成本书核心内容的第4个知识单元,主要围绕承担计算机的输入输出功能的设备或者部件进行讲解,给出了用于连接计算机各个部件的单总线结构的实际例子,具体介绍了串行接口的内部线路组成和使用方法,并通过串口连接PC仿真终端,选用程序查询方式控制入出设备,使教学机整机系统具备了输入输出操作功能。

第12章的流水线技术和第13章的并行计算机体系结构共同构成本书核心内容的第5个知识单元,对应计算机体系结构课程的基础知识,针对提高计算机系统的性能,更多地强调基本概念、提出问题的思路和解决问题的方案,基本上止步于定性说明。

教学过程中,可以根据不同的课程安排和教学要求,合理分配教材中3部分内容的课时比例。针对把计算机组成和系统结构合并成一门课程的安排,教材第3～13章的内容都属于必学知识,建议教学学时安排为70～90。若只是用于计算机组成原理课程,计算机体系结构的内容另外开课,则只需讲解第3～11章中的知识,建议课内学时安排为60～70,另外安排约16个实验学时。

第2章用于复习先修课程的内容,简明介绍数字电路与逻辑设计知识,约占教材总篇幅的7%,是学习计算机组成和体系结构一定会用到的电路基础知识,也许要求并不太多也不深,但如果完全不了解这些内容,要听懂课堂授课内容难度很大,设计实现一个小计算机系统更无从谈起。

第1章、第3～11章是课程的主体部分,约占教材总篇幅的73%,主要是计算机组成方

面较为完整的系统知识，重点围绕基本计算机硬件系统 5 个功能部件的功能和组成进行分析讲解。

第 12 章和第 13 章是本课程的提高部分，约占教材总篇幅的 20%，主要是计算机体系结构方面的基础知识，重点介绍提高计算机系统性能的各种可行思路与基本途径。

在教学环节安排中，需要处理好理论教学和教学实验的关系，可以考虑（并非一定如此）用约四分之三的课内学时（例如 48 学时）讲授计算机组成与运行机制的核心知识，四分之一的课内学时中的一小部分（例如 6 学时）用于讲解构建整机系统用到的设计技术和工程实现问题。剩余的部分（例如 10 学时）和 16 个实验学时统一安排用于教学实验，在教师的指导下去完成设计实现小计算机硬件系统的核心工作，更好地贯彻理论指导实践，通过实践再进一步深入理解理论的认知过程，做到学习知识和增长能力的双丰收。

本教材配套的有：①内容详尽的教学实验指导教材；②教学实验设备（由清华大学科教仪器厂生产销售，型号是 TEC-XP-Ⅱ），选用教材第 1～4 个知识单元的部件实例组合而成，能够确保课堂授课内容和教学实验项目完美的结合；③PowerPoint 教学课件；④指令级软件模拟系统，可以直接在 PC 系统中运行，实现了与硬件设备相同的运行功能。良好的教学实践环境和实验条件，可以有效地加深对课堂教学内容的理解，并使得学生在一定程度上获得开展研究工作和开展计算机硬件系统设计的实际经验，全面提高解决实际问题和创新思维的能力。

本书的第 1～3 章、第 7～13 章由宋佳兴修订，第 4～6 章由王诚修订，作者有多年从事本专业教学和科研工作的经历。

由于时间和作者水平所限，加上时间仓促，书中难免存在不足之处，敬请读者批评指正。

编　者

2016 年 6 月于清华大学计算机科学与技术系

目　录

CONTENTS

第1章 计算机系统概述

本章首先介绍计算机系统的基本组成和它的层次结构，使读者从层次的观点，初步认识完整计算机系统硬件与软件的基本组成，重点集中到计算机硬件的5个功能部件各自分担的功能及其相互的连接关系。接下来初步讨论计算机系统主要的性能和技术指标。之后对计算机硬件子系统的3部分知识，即计算机的体系结构、计算机组成和计算机实现进行说明，指明它们之间的联系与区别，帮助读者把握学习本门课程的主脉络。最后是计算机的发展过程，计算机系统的分类和推广应用的状况。

本章作为学习计算机组成原理和体系结构课程的引导性的提纲，介绍了计算机系统中的某些基本概念和常用术语，希望读者能够从硬件和软件、整机和部件、知识和能力等多种对应关系的角度来提高自己的学习效率。

1.1 计算机系统的基本组成及其层次结构

计算机系统(Computer System)是指电子数字通用计算机系统，3个定语各自表明了计算机系统的一个方面的特性。“电子”一词表明使用电子线路(不同于机械、继电器等)来实现计算机硬件的关键逻辑功能；“数字”一词表明使用的电子线路是数字式电路(不同于模拟电路)，运算和处理的数据是二进制的离散数据(不同于连续的电压或者电流量)；“通用”一词表明计算机本身的功能多样性(不是专用于某种特定功能)，具有完成各种运算或事物处理的能力。

完整的计算机系统由硬件(Hardware)和软件(Software)两大部分(两类资源)组成。计算机的硬件系统是计算机系统中看得见、摸得着的物理设备，是一种高度复杂的、由多种电子线路、精密机械装置等构成的、能自动并且高速地完成数据计算与处理的装置或者工具。计算机的软件系统是计算机系统中的程序和相关数据，包括完成计算机资源管理、方便用户使用的系统软件(厂家提供)和完成用户对数据的预期处理功能的用户软件(用户设计并自己使用)这样两大部分。软硬件二者相互依存，分工互动，缺一不可，硬件是计算机系统中保存与运行软件程序的物质基础，软件则是指挥硬件完成预期功能的智力部分，正如同一个健全和健康的人一样，必须同时具备物质性的肉体和精神性的智力与思维。

若进一步深入分析，还可以通过6个层次来认识计算机硬件和软件系统的组成关系，如图1.1所示。最下面的2层属于硬件内容，最上面的3层属于软件内容，中间的指令系统层

连接硬件和软件两部分,与两部分都有密切关系。

从图1.1中可以看出,计算机系统具有6层结构。在不同层次之间的关系表现为以下两个方面。

(1) 处在上面的一层是在下一层的基础上实现出来的,它实现的功能更强大,也就是说,更接近人解决问题的思维方式和处理问题的具体过程,对使用人员更方便,使用这一层提供的功能时,不必关心其下一层的实现细节。

(2) 处在下面的一层是实现上一层的基础,更接近计算机硬件实现的细节,它实现的功能相对简单,人们使用这些功能更感到困难。在实现这一层的功能时,可能尚无法全面了解其上一层的最终目标和将要解决的问题,也不必理解其更下一层实现中的有关细节问题,只要使用下一层所提供出来的功能来完成本层次的功能处理即可。

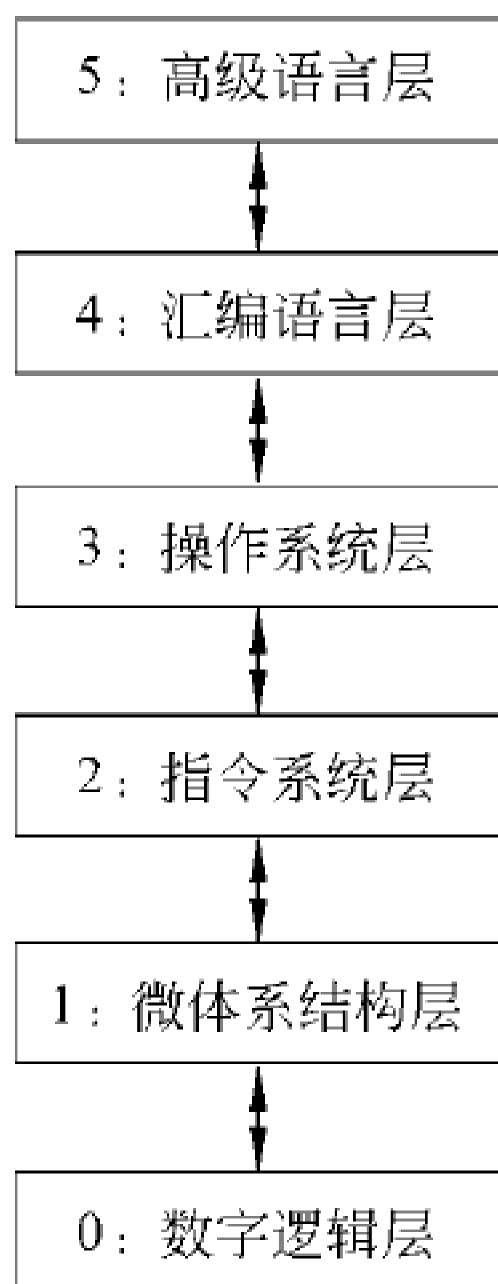


图1.1　计算机系统层次结构

采用这种分层次的方法来分析和解决某些问题,有利于简化处理问题的难度,在某一段时间,在处理某一层中的问题时,只需集中精力解决本层最需要关心的核心问题即可,而不必牵扯各上下层中的其他问题。例如,在用高级语言设计程序时,无须深入了解各低层的内容。

(1) 第0个层次是数字逻辑层,着重体现实现计算机硬件的最重要的物质材料是电子线路,能够直接处理离散的数字信号。设计计算机硬件的基础知识就是数字逻辑和数字门电路,解决的基本问题包括:使用何种线路和如何存储信息,使用何种线路和如何传送信息,使用何种线路和如何运算与加工信息等方面。这一部分属于计算机组成原理预备性的知识。

(2) 第1个层次是微体系结构(Micro Architecture)层,也称其为计算机裸机。计算机的核心功能是执行程序,程序是按一定规则和顺序组织起来的指令序列。这个层次着重体现的是,为了执行指令,需要在计算机中设置哪些功能部件(例如存储、运算、输入和输出、接口和总线等部件,当然还有更复杂一点的控制器部件),每个部件具体如何组成和怎样运行,这些部件如何实现相互连接并协同工作等方面的知识和技术。计算机硬件系统通常由运算器部件(数据通路)、控制器部件、存储器部件、输入设备和输出设备这5个部分组成,这些部分是计算机组成原理的主要内容。

(3) 第2个层次是指令系统(Instruction Set)层,介于硬件和软件之间。它涉及需要确定使用哪些指令;指令能够处理的数据类型和对各种类型数据可以执行的运算;每一条指令的格式和实现的功能;如何指出想要对其执行读操作或者写操作的存储器的一个存储单元;如何指出想要执行输入或者输出操作的一个外围设备;对哪些数据进行运算,执行哪一种运算;如何保存计算结果等。指令系统是计算机硬件系统设计、实现的最基本和最重要的依据,与计算机硬件实现的复杂程度、设计程序的难易程度、程序占用硬件资源的多少、程序运行的效率等都直接相关。也就是说,硬件系统就是要实现每一条指令的功能,能够直接识别和执行由指令代码组成的程序。当然,指令系统与计算机软件的关系也十分密切,指令是用于设计程序的。节省硬件资源和有利于提高程序运行效率是对指令系统的主要要求。在计

算机内部，全部的程序最终都由指令系统所提供的指令代码组成，计算机硬件能够直接识别和执行的只能是由指令代码组成的程序。一台计算机的指令系统对计算机厂家和用户来说都是很重要的事情，需要非常认真仔细地分析和对待。指令系统设计属于计算机系统结构的范围，合理选择可用的线路实现每一条指令的功能则是计算机组成的主要任务。

(4) 第3个层次是操作系统(Operating System)层。它是计算机系统中最重要的软件，主要负责计算机系统中的资源管理与分配，以及向使用者和程序设计人员提供简单、方便、高效的服务。一套计算机系统中包含了大量高价的、管理和使用相当复杂的硬件资源和软件资源。不仅一般水平的使用人员，就是水平很高的专业人员都难以直接控制和操作，因此把资源管理和调度功能留给计算机系统本身来完成更可靠，这些功能是由操作系统承担的。操作系统还为使用计算机的用户提供了许多支持，它与程序设计语言相结合，使得程序设计更简单，创建用户的应用程序和操作计算机也更方便。它是使用(直接或者间接)计算机指令系统所提供的指令设计出来的程序，并把一些常用功能以操作命令或者系统调用的方式提供给使用人员。也可以说，操作系统进一步扩展了原来的指令系统，提供了新的可用命令，从而构成一台比起纯硬件系统(计算机裸机)功能更加强大的计算机系统。操作系统不属于计算机组成的范围，在计算机专业的教学安排中应该设置这门软件课程。

(5) 第4个层次是汇编语言(Assembly Language)层。计算机是由人指挥控制，供人来使用的电子设备。使用计算机的人员要有办法把自己的意图告知给计算机，为完成这种“对话”，就需要使用某种语言。遗憾的是，计算机还不能(至少目前尚不能低成本的实现)听懂人类的自然语言，更无法执行人类自然语言的全部命令。最简单的解决办法是让计算机使用它的硬件可以直接识别、理解的，用电子线路容易处理的一种语言，这就是计算机的机器语言，又称为二进制代码语言，也就是计算机的指令。一台计算机的全部指令构成该计算机的指令系统(Instruction Set)。由此可以看出，计算机的基础硬件实质上是在机器语言的层次上被设计与实现出来的，并且可以直接识别和执行的只能是由机器语言构成的程序。这样做的结果是，计算机一方的矛盾解决了，但是使用计算机的人员却很难接受并使用这种语言。为此，必须找出一种折中方案，做到人员使用和计算机实现都更容易一点，这就要用到汇编语言和高级程序设计语言以及各种专用目的的语言。

汇编语言(Assembly Language)大体上是对计算机机器语言的符号化处理的结果，再增加一些为方便程序设计而实现的扩展功能。与机器语言相比，汇编语言至少有两大优点。首先实现用英文单词或其缩写形式替代二进制的指令代码，更容易为人们记忆和理解；其次是可以选用含义明确的英文单词来表示程序中用到的数据(常量和变量)，并且避免程序设计人员亲自花费精力为这些数据分配存储单元，而是将这些工作留给汇编程序自己去安排，这样的语言就达到了实用的最基本的标准。如果在此基础上，再支持程序的不同结构特性(如循环和重复执行等结构)，将子程序所用哑变元替换为真实参数等方面提供必要的支持，使用这个语言设计程序就更为方便。因此，汇编语言是面向计算机硬件本身的、程序设计人员可以使用的一种计算机语言。汇编语言的程序必须经过一个叫作汇编程序的系统软件的翻译，将其转换为计算机的机器语言后，才能在计算机的硬件系统上予以执行。由于汇编语言和机器语言存在十分紧密的对应关系，在讲授和学习计算机组成原理课程时，通常应使用汇编语言来设计实例程序。

(6) 第5个层次是高级语言层，高级语言又称算法语言(Algorithm Language)，它的实

现思路不再是过分地“靠拢”计算机硬件的指令系统,而是侧重面向解决实际问题所用的算法,更多的是为方便程序设计人员写出自己解决问题的处理方案和解题过程的程序。目前常用的高级语言有C、C++、PASCAL、Java、BASIC等。用这些语言设计出来的程序通常需要经过一个叫作编译程序的软件将其编译成机器语言程序,或者首先编译成汇编程序后,再经过汇编操作后得到机器语言程序,才能在计算机的硬件系统上予以执行。也可以由一个叫作解释执行程序的软件逐条取来相应高级语言程序的每个语句并直接控制其执行过程,而不是把整个程序编译为机器语言程序之后再交给硬件系统加以执行,解释执行程序的最大缺点是运行效率很低。高级语言不属于计算机组成课程的范围。

在高级语言层之上,还可以有应用层,由解决实际问题的处理程序组成,例如文字处理软件、数据库软件、网络软件、多媒体信息处理软件、办公自动化软件等。但这些内容已经超出了本书的讨论范围,不在这里赘述,换句话说,计算机是用于解决各种应用问题的系统,为有应用而存在,通过处理各种应用问题而体现出它的性能和价值。

在大部分的教材中,人们通常把没有配备软件的纯硬件系统称为“裸机”,这是计算机系统的根基或称“内核”。它的设计目标更多地集中到方便硬件实现和有利于降低成本两个方面,因此提供的功能相对较弱,只能执行由机器语言构成的程序,非常难以使用。为此,人们期望能开发出功能更强、更接近人的思维方式和使用习惯的语言,这是通过在裸机上配备适当的软件来完成的。每加一层软件就构成一个新的“虚拟计算机”,功能更强大,使用也更加方便。例如,配备了操作系统之后,就可以通过操作系统的命令(Command)或者窗口上的图标方便地操作使用这个新的虚拟机系统;再配备上汇编语言,用户就可以用它来编写用户程序,实现用户预期的处理功能;配备了高级语言之后,用户就可以用它来更方便高效地编写程序,解决处理规模更为庞大、逻辑关系更加复杂的问题。例如,可以把前面说明的计算机系统中的第1~5层分别称为裸机、L_1虚拟机(支持机器语言)、L_2虚拟机(增加了操作系统)、L_3虚拟机(支持汇编语言)、L_4虚拟机(支持高级语言)。

1.2 计算机硬件的5个功能部件及其功能

计算机系统的核心功能是执行程序。为此,首先必须有能力把要运行的程序和用到原始数据输入到计算机内部并存储起来,接下来应该有办法逐条执行这个程序中的指令以完成数据运算并得到运算结果,最后还要可以把运算结果输出供人检查和使用。为此,一套计算机的硬件系统至少需要由下述几个相互连接在一起的部件和设备组成,如图1.2所示。

在图1.2中通过5个方框图给出了计算机硬件的5个基本功能部件。其中的4个部件所分担的功能,通过方框中的文字说明已经表示出来。例如,数据输入设备完成对程序和原始数据的输入功能,数据存储部件完成对程序和数据的存储功能,数据运算部件完成对数据的运算处理功能,结果输出部件完成对运算处理结果的输出功能。而控制器部件则是依照每条指令的运行功能的需要,向各个部件或设备提供它们协调运行所需要的控制信号,在整个硬件系统中起到指挥、协调和控制的作用。

可以把计算机想象为一个加工、处理数据的工厂,则数据运算部件就是数据加工车间;数据存储部件就是存放原材料、半成品和最终产品的库房;输入设备相当于运入原材料的运货卡车;输出设备相当于发出最终产品的运货卡车;控制部件则相当于承担领导指挥功能的

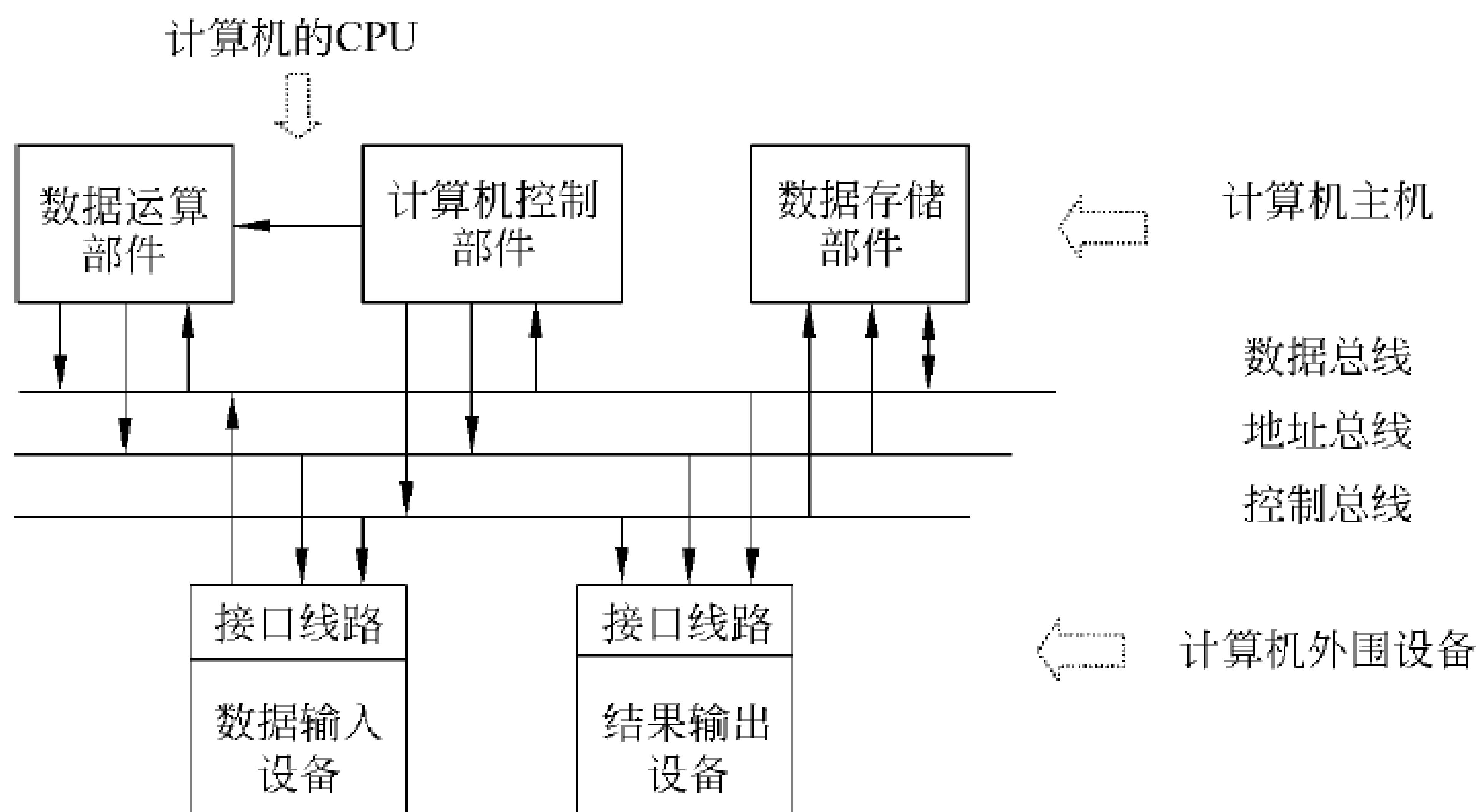


图 1.2　计算机硬件系统的组成示意图

厂长和各个职能办公室。在领导的正确指挥下，如果能够源源不断地取得原材料，工厂内又有存放的场所，车间能够对这些原材料进行指定的加工处理，加工后的产品可以畅通地运出去销售，则这个工厂（计算机）就纳入正常运行的轨道。

图 1.2 中上面标记为部件的 3 个组成部分通常是使用电子线路实现出来的，安装在一个金属机柜内或者印制电路板上，被称为计算机的主机。左边的数据运算部件和计算机控制部件合称为计算机的中央处理器（Center Processing Unit，CPU），又称为处理机（Processor）。

图 1.2 中下面标记为设备的 2 个组成部分通常是使用精密机械装置和电子线路共同制作出来的，也可以合称为输入输出设备，又称为计算机的外围设备。

图 1.2 中的中间是计算机的 3 种类型的总线。数据总线用于在这些部件或设备之间传送属于数据信息（指令和数据）的电气信号；地址总线用于在这些部件或设备之间传送属于地址信息的电气信号，以选择数据存储部件中的一个存储单元，或者外围设备中的一个设备；控制总线用于向存储部件和外围设备传送起控制作用的电气信号，也就是指定在 CPU 和这些部件或者设备之间数据传送的方向以及操作的性质（读操作还是写操作）等。请注意，CPU 和输入输出设备通常并不是直接通过总线相互连接起来，需要经过一个叫作接口的电路完成连接，这有利于降低 CPU 和设备之间的耦合强度，增强整机系统构成的灵活性。可以看出，计算机的 5 个功能部件正是通过这 3 种类型的总线以及接口被有机地连接在一起，从而构成一台完整的、可以协调运行（执行程序）的计算机硬件系统。

在计算机中，普遍的体系结构是由冯・诺依曼先生提出来的被称为存储程序的计算机，即可以把指令和数据用二进制代码形式保存到存储器中，并通过地址顺序访问。计算机由承担数据运算、数据存储、输入和输出、整机系统控制等功能的部件组成。早期计算机的几个部件是围绕着运算器部件来组织的，如图 1.3(a)所示方案。其特点是在存储器和输入输出设备之间传送数据都需要经过运算器。在当前流行的计算机中，更常用的方案则是围绕着存储器部件来组织，如图 1.3(b)所示方案。方案(b)和方案(a)相比，并无实质性的区别，只是在一些小的方面做了部分改进，使输入输出操作尽可能地绕过 CPU，直接在输入输出设备和存储器系统之间完成，以提高系统的整体运行性能。

前面说到的还只限于“工厂的硬件”组成，也就是人员和厂房设备等，仅有这些，工厂还

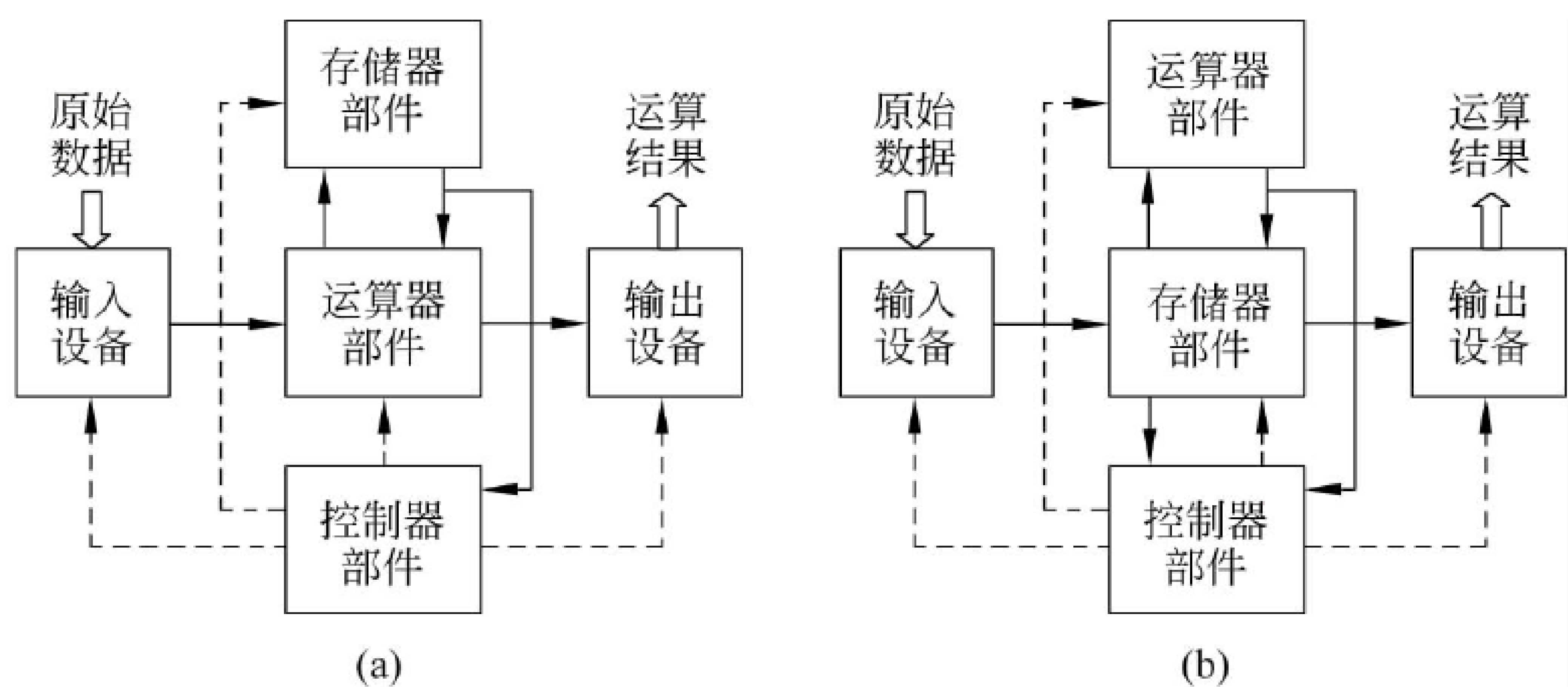


图 1.3　计算机的组成结构

是运转不起来的,至少是很难运行的。要成功运转,还需要有一系列的规章制度、管理策略和经营办法等“软件”部分。计算机系统也一样,在硬件组成的基础之上,还必须有它的软件部分。其主要包括操作系统、程序设计语言及其支持软件等,这些在前面已经提到。

如表 1.1 所示,以计算从 1 到 10 的累加和为例,看一看 3 个级别语言的程序例子,并简单介绍该机器语言程序在计算机内部的执行过程。下面的程序可以在一个模拟软件中实际运行。

表 1.1　3 个级别的语言比较

高级语言 BASIC 的程序	汇编语言的程序	机器语言的程序(十六进制表示)
10 sum=0	2000: sub R15, R15	01FF
20 for i=1 to 10	sub R1, R1	0111
30 sum=sum+i	mvrd R0,0A	8800 000A
40 next i	inc R1	0910
50 print sum	add R15,R1	00F1
60 end	cmp R1, R0	0310
	jrnz 2004	47FC
	cala 0664	CE00 0064
	ret	8F00
每个 BASIC 语句带数字标号,用于表明语句次序关系,用高级语言设计程序,重点关注运算功能而不是计算机硬件组成及使用,很方便	设计汇编语言程序要懂得计算机硬件的某些特性,要自己安排通用寄存器和内存地址等,比较烦琐、复杂,工作效率低	机器语言程序是计算机指令代码的序列,很难记忆指令代码并用其设计程序,每个基本汇编语句对应一条指令

机器码程序的执行过程为:首先把机器语言的程序用输入设备输入到计算机的存储器中,例如从十六进制的 2000 地址单元开始依次存放每一条指令。用操作系统(监控程序)的命令运行这个程序,则 CPU 将从程序的首地址(2000)到存储器读来第一条指令并保存到控制器中,接下来依据指令功能指挥各部件完成必要操作。例如第 1 条就是控制运算器完成一次减法运算,与此同时形成下一条指令的地址,待本条指令完成后,用这个地址去读下一条指令,继续执行,直到程序的最后一条指令。cala 语句调用子程序,把 R15 中的累加和显示到屏幕上。

1.3 计算机系统主要的技术与性能指标

这里只是从整机的角度介绍计算机系统的某些技术与性能指标，突出几个重要概念和基本术语，而把涉及各个部件的更具体的指标安排到其他相应的章节中介绍。

1. 计算机字长

从物理上容易实现和数据运算规则简单进行考虑，现代的计算机普遍使用二进制，即每一位上的数值只有 0 和 1 两个值，相邻数位之间采用“逢二进 1”的规则处理，用从右到左依次排列起来的一串二进制的数表示不同的数值和信息。

在计算机系统内部，通常选用多少个二进制位来表示一个数据或一条指令是一个关键技术指标，例如 16、32 或者 64 位，这个位数被称为计算机字长。现代计算机的字长通常都被设计为 8 的整数倍。如在 32 位字长的计算机系统中，一个整数、一条指令通常都用 32 位二进制数表示，叫作一个字，运算器、存储器、数据和地址总线等通常都被设计成 32 位。字长对计算机的处理能力和运行性能有明显影响，字长较长有利于提高计算机的性能，但需要使用更多的硬件，计算机系统的价格也会高一些。

2. CPU 速度

衡量 CPU 速度，通常有两种方式。

第一种方式是使用 CPU 主频，即 CPU 系统使用的时钟脉冲的频率（每一秒提供的时钟脉冲的个数称为赫兹（Hz），10^6 Hz=1MHz，10^9 Hz=1GHz）来表示，例如 500MHz。对同一个型号的计算机，其主频越高，完成指令的一个执行步骤所用的时间越短，执行指令的速度越快。但对不同厂家、不同系列的计算机系统，只用 CPU 主频来说明其运行速度则未必准确。

第二种方式是使用 CPU 每一秒能执行的指令条数，单位是 MIPS（Million Instructions Per Second），其计算公式可以通过如下方式推导出来。

$$T = \text{CPI} \times \text{T_IC} \times I$$

这里的 T 是执行一个程序占用的全部时间，CPI 是执行一条指令平均使用的 CPU 时钟个数，T_IC 是一个 CPU 时钟的时间长度，是 CPU 主频 f 的倒数 $1/f$，I 是这个程序的指令条数，3 个数值的乘积就等于这个程序总的运行时间 T。由此得到：

$$I = T/(\text{CPI} \times 1/f) = T \times f/\text{CPI}$$

这个公式表明，单位时间内执行的指令条数正比于 CPU 的时钟频率 f，这个频率的高低取决于计算机的实现技术、生产工艺和计算机组成；反比于每条指令的执行步骤数目。它反映计算机的实现技术、计算机指令系统的结构和计算机组成；一个程序的指令条数还与计算机指令系统的结构和编译技术有关系。

当取 T=1s，并假定 f 为 300MHz，CPI 为 4，则计算出该 CPU 系统的性能为 300/4，等于 75 个 MIPS，即每秒执行 75 个百万条指令。若有办法使这台计算机的 CPI 尽量靠近 1，则其运行性能就可以提高近 4 倍，这正是精简指令系统计算机（RISC）所追求的目标。

若进一步细化，可以写出计算 CPI 的公式为 $\text{CPI} = \sum_{j=1}^{n} \text{CPI}_j \times I_j/I$，这里的 n 是指令的种类，CPI_j 是每类指令的执行步骤数，I_j/I 表示在程序中这类指令数目与总指令数目的比

例。这种通过引入不同指令在程序中出现的频率来计算加权CPI的方法，更符合计算机系统运行的真实性能。

由于在计算机中用于计算整数的指令和计算浮点数(实数)的指令执行速度差异较大，不同程序中这两类指令所占的比例也有很大不同，所以在许多场合下，人们分别用MIPS和MFLOPS描述整数指令和浮点数指令的执行速度，以对比不同计算机系统的CPU性能水平。

3. 存储容量

计算机中的存储器通常包括内存储器和外存储器两大类。内存储器又被称为主存储器，通常用半导体器件实现。其读写速度快，价格较高，通常容量要小一些，可供CPU通过指令直接访问。断电后随机读写的存储器(RAM)原保存的信息将丢失，只读存储器(ROM)中的信息将保持不变。外存储器又称辅助存储器、海量存储器等，主要包括磁盘设备、光盘设备、磁带设备等。通常是在机械旋转或移动的盘片、磁带上设置一层记录信息的物质，用磁化、改变反射光强度的方式写入或读出二进制的信息，读写速度要慢得多，价格较低，容量很大。外存储器上的信息需要经过操作系统成批量地(而不是以字为单位)与内存储器进行交换。外存储器中的信息在断电后不会丢失，在脱机的状态下也可以长期保存。存储器的容量，通常情况下用字节(Byte)表示，32位的字由4个字节组成。存储器的容量大，所存储的信息量就越大，计算机运行的速度就可能更快，相应的硬件成本也会更高。

目前计算机的内存容量为几十、几百兆字节，甚至几十、上百吉字节。外存储器的容量一般是几十、几百吉字节，采用阵列技术则可以得到容量更大的(例如几十、上千太字节，$1TB=10^{12}B$)外存储器系统。

4. 内存储器的存取周期和外存储器的数据传送速率

内存储器的读写周期是指启动连续的两次读写操作所必需的时间间隔，通常都比较短。当前的内存储器存取周期为几十个纳秒到一两百个纳秒(ns)，读写速度快的存储器价格要贵一些，要合理选用。外存储器会涉及机械运动，找到要读写的数据在硬磁盘中的位置通常需要几十到十几个毫秒(ms)，一次读出几百个字节的信息通常需要若干毫秒，而连续读出一批数据平均到每个信息上的读出时间可以更短。通常把单位时间内可以对磁盘设备读写的数据数量称为设备的传输速率，例如磁盘设备的传输速率可以达到几十到几百兆字节每秒(MB/s)。外存设备的传输速率是影响计算机系统性能的一个重要因素，特别是作为服务器之类的计算机更是如此。这个速率不仅与设备本身的性能有关，与设备的接口性能和计算机总线设计也直接相关。在使用磁盘阵列技术之后，其传输速率可以得到大幅度提高。

5. 输入输出设备的入出速度

计算机的输入输出设备是计算机系统中比较复杂的部分。其组成和运行原理各不相同，与计算机主机的连接与控制方式也有很大差异。在电子线路之外还涉及精密机械、光学、激光、电磁转换等许多知识。不同的输入输出设备的运行速度各不相同，例如针式打印机每秒只能打印几个字符，而激光打印机则可以打印多行甚至几页打印纸的信息，键盘输入则主要取决于人员打字的速度。显示器的屏幕大小、分辨率高低、显示字符还是图形的不同内容、屏幕刷新频率等都对系统性能有重要影响。

通常还需要关注**计算机内部的并行处理能力**。一个部件通常一次只能完成一项功能，若同时设置两个部件，则它们就可以同时各自完成一项功能，这被称为并行处理，是提高计

算机系统性能的重要途径之一。例如,运算器中可以设置两个完成整数运算的定点运算器,控制器可以控制多条指令并行执行;内存储器可以被划分成多个可以并行读写的存储体;可以使用多个可以并行读写的磁盘构成一个磁盘阵列;在一个计算机系统中设置多个 CPU;使用多台计算机系统协同执行一项大型的计算任务等。也可以通过分时的办法使几个软件程序并行运行。软硬件的并行处理能力是计算机系统结构重点研究的问题,会反映到计算机组成中来,但不属于计算机组成原理课程的基本教学内容。

1.4 计算机的体系结构、组成和实现概述

在讲解计算机硬件系统时,通常会涉及计算机体系结构、计算机组成和计算机实现这 3 个既有联系又有区别的不同概念。它们各自处理和解决的问题不尽相同,下面对此进行简要介绍,并结合设计实现教学实验计算机系统处理的某些问题做一些补充说明。

计算机体系结构(Computer Architecture)通常是指使用机器语言或者汇编语言的程序设计人员所见到的计算机系统的属性,即计算机的使用特性,是计算机系统的概念性结构及其功能特性。这其中的许多问题都直接和计算机的指令系统有关,例如计算机的字长,指令类别、格式和功能,支持的寻址方式,指令系统的构成,计算机硬件能够直接识别和处理的数据类型及其表示、存储、读写方式,指令中使用的寄存器数量和表示方法,存储器、输入输出设备和 CPU 之间数据传送的方式和控制,也包括中断的类型和处理流程,系统中对各类信息的保护,计算机的运行状态的定义和切换,对各种运行异常或者出错的检测和处理方案等。这些都是程序设计人员编写出高质量程序并确保其正常运行必须深入了解的计算机的属性。计算机体系结构主要研究硬件和软件功能的划分,确定硬件和软件的界线,即哪些功能应划分给硬件子系统完成,哪些功能应划分到软件子系统中实现。

计算机组成(Computer Organization)是在确定了硬件子系统的概念结构和功能特性的基础上,设计计算机各部件的具体组成、它们之间的连接关系,实现机器指令级的各种功能和特性。从这一点又可以说,计算机组成是计算机体系结构的逻辑实现。为了实现相同的计算机体系结构所要求的功能,完全可以有多种不同的计算机组成设计方案。半导体器件性能的提高,新的技术成果的面世,新的价格和性能比的需求出现,都会带来计算机组成的变化。在这个方面最突出的例子,就是系列计算机(Family Computer)的出现和它的极大成功。例如,IBM 的 360 系列和 370 系列计算机,都由一系列的性能不同、价格不同、具体组成和实现也不同的机型组成,但它们却具有相同的指令集,也就是说,同一个机器语言程序可以不加修改地在这些不同机型的机器中运行,至少在后来出现的性能更高的新机型上能运行在早期机型中使用的已有程序(反过来则未必行得通),这被称为系列机中的软件兼容(Software Compatibility),这对于保护计算机厂家和计算机用户的软件投资都是十分重要的。另外还出现了硬件、软件的“第三方产品”,一些规模较小的厂家,遵照知名厂家的计算机体系结构,或者软件产品功能,自己重新设计并实现具有更高性能价格比的(用户更欢迎)新的计算机组成,或者更廉价的软件产品,以便在知名大公司的“夹缝”中找到一线生存空间,这也是推动计算机技术进步的一种额外动力。还是再次回到计算机组成这一问题本身上来。在计算机组成的领域内,需要重点解决的问题之一是合理的性能价格比。其关键的技术措施在于处理好计算机内部的数据流和控制流,合理地匹配各功能部件的性能参数,也

就是尽力避免因一个部件形成的“瓶颈”问题而影响计算机的整体性能。例如,对运算器部件,可以通过实现数据运算的流水线处理和设置多个运算功能部件,在运算器内安排更多的寄存器等措施以提高其处理数据的能力;对控制器部件,可以通过指令预取,指令流水线处理,多指令流水线,选用 RISC(Reduced Instruction Set Computer)结构设计方案等措施以提高执行指令的速度;对存储器部件,使用由高速缓冲存储器、主存储器、虚拟存储器构成的层次结构的存储系统,使用由可以交替运行的多个存储器构成的多体结构,使用性能更高的改进型的存储器芯片等措施,以提高存储器系统的存储容量和读写速度。对输入输出设备,实现通道、外围处理机等方式,合理地设置缓冲器和排队策略,配备速度更快的设备,配备更多数量的设备,以提高单位时间内数据输入输出的流量。对计算机系统而言,关键是尽可能地使计算机各个功能部件都以自己所具有的高速度运行,避免或者减少不同功能部件彼此之间的相互制约和等待现象。例如,通过支持多线程、多进程、多道程序、多任务等措施,选用最合理的资源调度算法和分配策略,以便最大限度地提高系统的资源利用率。

计算机实现(Computer Implementation)是计算机组成的物理实现。其包括中央处理机、主存储器、输入输出接口和设备的物理结构,所选用的半导体器件的集成度和速度,器件、模块、插件、底板的划分,电源、冷却、装配等技术,生产工艺和系统调试等各种问题。可以用一句话概括,就是把完成逻辑设计的计算机组成方案转换为真实的计算机设备,也就是把满足设计和运行、价格等各项要求的计算机系统真正地制作并调试出来。

计算机体系结构、计算机组成和计算机实现是3个不同的概念,各有不同的含义,但是又有着密切的联系,而且随着时间和技术的进步,这些含意也会有所改变。在某些情况下,有时也无须特意地去区分计算机体系结构和计算机组成的不同含义。

若对上述3项内容的讲解止步于此,就难以给学生留下明确印象,我们还是结合研制开发教学计算机的完整过程,看一下有哪些问题应该想到并需要解决,这部分内容仅用于开阔同学的视野,不属于基本教学内容。首先概述教学机的规划目标、关键技术、实施措施,接下来从教学计算机的系统结构、组成、实现3个角度列举各自要处理的部分问题。

1. 规划课题目标,明确产品特点

(1) 设计实现一台用于计算机硬件课程教学的计算机系统,要求硬件组成简单完整,体现原理清楚,配备监控程序,支持汇编编程,实验功能强大,方便操作运行。

(2) 能实时显示整机系统内部的信息存储与传送的时间—空间关系,是对教学实验设备的特殊要求,这对学生学懂计算机组成原理及其运行机制、深入理解实验内容至关重要。

2. 选择设计技术,确定技术方案

(1) 控制器选用现场可编程的 isp MACH 器件实现,并把另外一些功能电路也纳入到该芯片之中;运算器、内存和串行接口选用中小规模电路实现,其中的运算器使用4片4位位片结构的运算器芯片实现,内存选用静态存储器芯片并使用字位扩展技术实现,与 CPU 以同步方式运行。

(2) 输入输出设备选用计算机仿真终端,使用串行接口把仿真终端接入主机系统,输入输出采用程序直接控制方式完成;整机系统采用单总线结构,即功能部件之间都通过这组总线传送信息。

(3) 选用硬件描述语言 ABEL-HDL 的逻辑方程方式和真值表方式描述 MACH 芯片内部的电路结构及其实现的功能。

（4）选用 Altium Designer 软件绘制教学机系统完整的原理图，并完成印制电路板上的元器件布局和布线。

3. 规划工程实现，确定必要措施

（1）采用一块双面印制电路板布放全部元器件，控制器中的 IR 插接到这块电路板上，其他 3 个子部件（PC、Timing、CU）设置在 MACH 芯片内。

（2）全部的半导体器件都通过器件插座插接在电路板上，方便系统调试与维修。

（3）实现硬布线和微程序两种控制器，使用两个 ABEL 程序分别描述其电路结构和功能。

（4）电路板上设置数量足够又可灵活选用的拨数开关和指示灯，并为数据总线 DB、地址总线 AB、指令寄存器 IR、节拍发生器 Timing 设置专用指示灯，以便随时查看计算机内部的数据、指令、状态信号、控制信号的当前值；在各部件之间的信息传送线上设置用于接线的插针，既可以连接通用的拨数开关，也可以连接通用的指示灯。

确定指令格式、功能和指令系统属于系统结构要处理和解决的问题，例如：

（1）指令字长 16 位，操作码占 8 位，单字指令为主，支持最常用的基本寻址方式。

（2）尽可能小的指令集，指令数目要适当地少。

（3）指令系统要有一定的完备程度，指令格式适当规范，有较好的典型性，指令分类合理，适当地靠近早期计算机的指令系统，但要大力简化。

（4）更高的可扩充性，即为学生添加各种新的指令留足余地，把指令划分为基本指令组（设计监控程序必用）和扩展指令组（由学生扩展，设计 BASIC 解释程序要用到）两类。

（5）满足教学计算机特定的实验要求，易于硬件实现，方便设计汇编和反汇编程序。

设计计算机的硬件系统组成属于计算机组成的内容，围绕如何实现每条指令的具体功能开展工作，需要处理好如下一些问题。

（1）硬件部件完整，包含计算机传统的 5 个功能部件，其中主存储器选用静态存储器芯片实现，要用到字位扩展技术，输入设备（键盘）和输出设备（显示器）可选用计算机仿真终端实现，要求使用串行接口芯片将其接入主机系统。

（2）配备小巧的软件系统，使用已有的基本指令设计监控程序，以类似计算机 Debug 程序的方式运行，支持汇编语言编程。

（3）监控程序以等待输入监控命令、分析命令、执行命令这 3 个步骤的循环方式运行，支持使用 G 和 T 命令调用用户程序，要正确处理监控程序和用户程序的现场切换，允许在用户程序中调用监控程序中的子程序。

把计算机系统制作出来并完成调试属于计算机实现的内容，要解决好如下一些问题。

（1）使用尽可能少的电路芯片构成运算器和存储器，整机系统制作在一块电路板上并合理布局，使部件划分和连接关系尽可能地简单清晰，这有利于突出计算机组成的原理知识，避免被过多的线路模糊了视线。

（2）选用较高集成度的一片可现场编程的器件实现控制器，把尽可能多的功能电路纳入到这个芯片中实现，提供先进的计算机设计技术，选用 ABEL 语言描述控制器的组成结构与功能实现，更有利于学生深入理解原理知识和简化实验操作。

（3）在设备的主板上，应以醒目方式区分计算机关键部件和实验辅助电路两个部分，后者是为支持各项教学实验而设置的，包括向系统内拨入调试信息用到的多组开关的元器件、

显示计算机内部信息状态用到的指示灯的元器件,以及时钟产生与启停控制电路等,这些不属于计算机组成的基本教学内容,会用即可;选用6组的8位开关向系统中的部件(芯片)提供输入信息,或选用4组的8位指示灯显示系统内部的有关信息都要通过接线操作实现;另外有3组16位的指示灯属于专用,分别接在数据总线DB、地址总线AB、指令寄存器IR的输出管脚,充分用好这3组指示灯对教学实验大有裨益。

(4) 为了支持使用6组拨数开关向某个部件或系统拨入信息,或者使用4组通用指示灯显示某些信息,需要在教学机的主要数据或信号的传送线上设置接线用的插针、插孔,这极大地提高了这些辅助电路的使用效率,也为不同层次的教学实验提供了更大的灵活性和选择空间,确保计算机内部信息的时间—空间关系都可以看到,可以有效提高教学实验的质量。

(5) 为了更简便地支持不同层次(芯片级、部件级、整机系统级)的教学实验,简化整机系统构建和部件拆分,在教学机主板上设置有4个小开关,分别用于接通Am2901芯片(运算器)、MACH芯片(控制器)、Am2910芯片(微程序定序芯片)、FPGA芯片(另外一个CPU)的电源,只向用到的芯片供电,关掉暂不用芯片的电源,则它就不再影响与它有连接关系的电路的运行状态,确保可以孤立出某个(些)芯片、某个(些)部件,可依据实验需求灵活安排。

在本教材中,我们把更多的注意力放到了计算机组成方面,例如第4章的运算器,第6到第11章的控制器、主存储器、高速缓存、虚拟存储器、输入输出系统、输入输出设备等是更多地围绕计算机组成原理的教学要求来编写的,突出知识的基础性和实用性。这部分知识和相关技术是计算机硬件的基础内容,也是学习和理解计算机体系结构的必要准备。把计算机体系结构的主要内容集中安排到第12章(指令流水线技术)和第13章(并行计算机体系结构),重点放到提高计算机性能的可行途径和有关技术。就计算机实现方面,我们强调可以使用计算机组成的基础知识及其部件实例构造出一台能正常运行的计算机系统,会涉及某些具体的工程与技术问题,将在配套的实验指教材中进行说明。

1.5 计算机发展进步、分类和拓展应用的进程

计算机原本的含义,简单地从字面上讲,指的是能够完成数值计算功能的工具。例如,在我国已经使用了2500多年、现在仍被使用的算盘就属于这一范围。它采用一竖列中的多个算盘珠子表示十进制数一位上的数值,为0~9,连续的几列表示十进制数不同的数位,位之间采用十进制规则,人们通过拨动算盘珠子,就能方便快速地完成加、减、乘、除等数值计算过程。在电子计算器或者电子计算机普及之前,计算尺也是工程界广泛使用的计算工具之一。计算尺是在木制的尺子形状的材料上,印上各种刻度和数字标记,通过拉动中间可移动的部分,找出不同位置上刻度(数字标记)的对应关系,来完成一次计算过程,在科学研究、工程设计等数值计算的领域曾被广泛应用,计算功能的专用性和较大的误差限制了它的应用范围。与此相呼应的,还有通过齿轮和拉动杆等做成的机械式计算机,包括手摇的或者电动的,也曾被广泛地用于完成加、减、乘、除等算术运算,通用性较好,但运算效率仍然比较低。

现在谈论的计算机,通常指的都是电子数字计算机系统。与前面刚提到的几种计算工

具相比，无论从它的通用性、计算速度、处理数据的能力、实现计算自动化的程度等任何一个方面去看，都有了本质性的变化。电子数字计算机诞生半个多世纪以来，其性能提高之快，应用领域拓展之宽，对社会发展以及人们的生活方式的影响之深远，令人震撼。

计算机系统的性能，主要指的是它的运行速度和处理数据的能力。不妨先简单地回顾一下计算机性能增长的历程。

(1) 20 世纪 70 年代之前，计算机性能增长速度缓慢，但形成了至今仍被广泛采用的冯·诺依曼先生提出的存储程序计算机的完整概念。

(2) 到了 20 世纪 70 年代，由于集成电路的出现和迅速发展，推动计算机的性能以每年 25％～30％的速度增长。

(3) 20 世纪 80 年代之后，集成电路技术的进步并结合计算机体系结构的变革，计算机的性能更达到了每年 50％的增长速度。

(4) 到了 20 世纪 90 年代中期之后，主要依靠计算机体系结构的发展，计算机的性能仍保持了每年 50％的增长速度。可以看到，计算机性能的提高，在物质的层面上，依靠的是集成电路生产工艺改进所带来的半导体器件性能的提高，在技术的层面上，依靠的是计算机体系结构和组成方面的创新与进步。

从制作计算机使用的元器件的不同来看，计算机的发展依次经历了电子管时代，晶体管时代，中小规模集成电路时代，大规模、超大规模集成电路时代 4 个不同的发展阶段。

(1) 电子管计算机时代。电子管是封装在玻璃外壳内的一种电真空器件，用它可以设计出实现反相功能的反相器线路，在此基础上，再设计出计算机所使用的全部组合逻辑线路，诸如加法器、译码器等线路，以及触发器、寄存器、计数器等各种时序逻辑线路。这时的计算机是用分立的元件、器件实现出来的。最早的一台电子管计算机包括 18000 多个电子管，1500 多个继电器，体积庞大，耗电和发热量也很大，价格极高，但实现的处理功能相对较低，每秒可以完成 5000 次的十进制加法运算，连存储器部件都没有，它的程序是通过插线和开关实现的。计算机可靠性也差，应用范围很窄。到后来，形成了存储程序计算机的概念，并用磁芯构建了计算机的存储器部件，程序、数据的输入和输出也可以通过纸带穿孔机设备、卡片机设备等完成。还制作出用于保存数据的磁带、磁鼓等外存储器设备，计算机开始进入了商业性应用阶段。

(2) 晶体管计算机时代。晶体管通常指的是晶体三极管，是用半导体材料制作出来、封装在一个金属壳内的带有 3 个管脚的小器件，1958 年进入批量生产阶段。用它可以设计出实现反相功能的反相器线路，在此基础上，再设计出计算机所使用的全部组合逻辑线路，以及触发器、寄存器、计数器等各种时序逻辑线路。这时的计算机仍然是用分立的元件、器件实现出来的，但与电子管的计算机相比，晶体管的体积更小，耗电和发热量更低，工作可靠性更高。因此，晶体管计算机体积要小得多，有条件实现更复杂和更强的处理功能，运行速度更快，价格上也有所下降，系统运行的可靠性得到明显提高，应用范围有了很大的拓展，计算机真正进入了广泛应用的阶段。IBM 公司经历了艰苦的创业过程，并初步登上了这一领域的霸主地位。

(3) 中小规模集成电路时代。随着半导体器件生产工艺与技术上的进步，在一片半导体基片上，可以生产出多个晶体管，并用它们形成具有一定处理功能的逻辑器件，这就是集成电路(Integrated Circuit)。此时集成到一个芯片内的晶体管数量还相当有限，实现的还

只限于简单的、完成基本处理功能的组合逻辑门一级的电路和简单的触发器、寄存器之类的电路,故被称为中小规模集成电路。使用这种器件设计与实现出来的计算机属于中小规模集成电路的计算机,它更精小和廉价,计算能力和可靠性也更高,为计算机的普及应用奠定了坚实的基础。在此期间,IBM 在其 System/360 大型计算机产品中,形成了计算机体系结构的基本概念,制作出有着相同体系结构却有着不同组成和不同性能的系列计算机,进一步捍卫了自己在计算机领域的世界霸主地位。

(4) 大规模和超大规模集成电路时代。半导体器件生产工艺的改进,使得在一片半导体基片上,可以生产出数量更多的晶体管,就形成了大规模集成(Large Scale Integration)电路;若在一个芯片上的晶体管数量达到更多,就被叫作超大规模集成(Super Large Scale Integration)电路;单个芯片内的晶体管数量达到百万个时被叫作甚大规模集成(Ultra Large Scale Integration)电路;达到一亿个时被叫作极大规模集成(Extremely Large Scale Integration)电路。用大规模和超大规模集成电路制作出来的计算机属于大规模、超大规模集成电路的计算机。其代表产品就是通过单片 IC 制作出计算机的 CPU,或者微处理机(Microprocessor)。个人计算机的出现和普遍应用是这一期间的重要成就。集成在一个芯片中的电路数量越多,就越能够提供出更强处理能力的计算机部件,甚至一个高性能的计算机处理器。更强处理能力和处理速度的工作站(Workstation)和精简指令系统的计算机(Reduced Instruction Set Computer,RISC)的出现,进一步推动了计算机体系结构和实现技术的发展。超标量技术(多发射指令技术)和乱序执行(Out-of-order Execution)的使用都是本时期的重要技术成果。

上面是根据实现计算机所选用电路的器件种类来划分计算机的发展进程的,依据这些器件出现的先后顺序又表现为电子管时代,晶体管时代,中小规模集成电路时代,大规模和超大规模集成电路时代。如果依据计算机所提供的功能和运行性能来划分,又可以把计算机划分为巨型计算机、大型计算机、中小型计算机和微型计算机,这种划分标准并不是绝对的,把一台计算机划分在哪一类中,和计算机发展的进程密切相关。例如,现在的一台微型计算机的功能和性能,可能比 40 年前的大型计算机的功能还要强大。这是计算机发展进步的重要成果和具体表现。

计算机另外一种分类办法是 Flynn 分类法。在 1966 年,M. J. Flynn 提出如下定义:

① 指令流(Instruction Stream)——机器执行的指令序列;

② 数据流(Data Stream)——由指令流调用的数据序列,包括输入数据和中间数据;

③ 多倍性(Multiplicity)——在系统中最受限制的元件上同时处于同一执行阶段的指令或数据的最大可能个数。

其按照指令流和数据流的不同的组织方式,把计算机系统的结构分为以下 4 类:

① 单指令流单数据流(Single Instruction stream Single Data stream,SISD);

② 单指令流多数据流(Single Instruction stream Multiple Data stream,SIMD);

③ 多指令流单数据流(Multiple Instruction stream Single Data stream,MISD);

④ 多指令流多数据流(Multiple Instruction stream Multiple Data stream,MIMD)。

上面的第 1 种是传统的指令顺序执行的处理机系统;第 2 种以阵列处理机或并行处理机为代表;第 4 种属于多处理机系统。多数人对第 3 种有不同的看法,认为这是一种难以实用的方案,在有的文献中把流水线结构的计算机看作为 MISD 结构。

计算机的发展、进步的另一个方面,表现在其应用领域的扩展和对社会变革、进步带来的深刻影响。简单列举(并非完整、全面)如下。

(1) 计算机作为计算工具,完成各种复杂的科学计算是它的一个重要应用方面。其在科学研究、工程设计、天气预报、地质与石油勘探等各个领域发挥着重要的作用。

(2) 计算机作为数据处理工具,在政府办公,企、事业单位的管理等领域也发挥着重要的作用。对政府大量的档案和公文收发、保存和检索,业务信息的处理与办公流程自动化管理,信息分析与科学决策等方面提供支持。对企业单位的人、财、物、购、销、存等信息的保存与管理,市场预测和经营决策等方面提供支持。

(3) 计算机作为具有高速和灵活的逻辑处理能力的工具,被广泛地应用于工业生产、航天发射等过程的实时控制。其包括用于产品的辅助设计(CAD)和辅助制造(CAM)。

(4) 计算机作为具有高速和灵活的逻辑处理和推理能力的工具,在人工智能领域,完成诸如数学定理证明,自然语言理解,知识表示和挖掘,密码处理和破译,智能机器人研制和应用,计算机翻译等需要有一定逻辑推理的领域发挥着重要的作用。

(5) 随着计算机网络的出现和发展,计算机已经成为在大范围内传播信息和实现人员沟通的重要工具。之后相继出现了电子政府、电子商务、网络教育、远程诊断、网上信息检索等多种应用,极大地改变了人类的生活环境和交流方式。

本章内容小结和学习方法建议

本章是作为学习计算机组成原理和体系结构课程的引导性提纲来编写的。其主要目标不是让读者掌握计算机组成与体系结构具体深入的知识,而是希望读者能初步理解计算机系统的总体构成及其关键术语,初步了解计算机系统中的主要技术与性能指标;理解计算机硬件的 5 个功能及其各自分担的功能,了解计算机系统结构和计算机组成各自要解决的主要问题。本章的要求是了解计算机系统中的硬件和软件组成及其层次关系,3 个级别的计算机语言;了解计算机的发展过程及其各个阶段的重要进展,计算机分类方法,计算机的应用领域及其对社会发展和人类生活的重大影响。这些内容虽然浅显,但对把握本课程教学的主脉络是有益的。在后续的学习过程中,不妨随时回过头来与这里的内容对照一下。

可以通过 1～6 这几个数字来概括本章的基本内容。

一个完整的计算机系统由硬件和软件 2 个子系统组成,它们被划分在 6 个不同的层次;硬件子系统由 5 个功能部件组成,要通过 3 种类型的总线连接在一起,涉及 5 项性能指标;软件子系统有 3 个级别的语言;可按照 Flynn 分类标准划分计算机为 4 种不同的结构。

习题与思考题

1. 什么是计算机系统?哪些部分属于硬件系统,哪些部分属于软件系统?应如何看待二者之间的关系?

2. 人们通常说的冯·诺依曼结构的计算机有哪些特点?

3. 从传统的观点来看,基本计算机硬件系统由哪几个功能部件组成?每个部件完成的主要功能是什么?它们之间是如何连接在一起的?

4. 什么是计算机系统的层次结构？一般可以划分为哪几个层次？层次之间的关系体现在哪些方面？采用层次结构的观点来看待、分析计算机系统的功能和组成有什么好处？

5. 指令系统对计算机硬件和软件的影响表现在哪些方面？

6. 通常可以把程序设计语言划分为哪几个层次？各自的优、缺点表现在哪些方面？

7. 在比较不同计算系统的性能时，通常可以从哪几个基本的技术与性能指标入手？

8. 计算机体系结构、计算机组成和计算机实现各自主要处理哪些方面的问题？你是怎样看待三者之间的关系的？

9. 计算机发展经历了哪几个时代？每个发展时代的计算机的主要特点表现在哪些方面？

10. 有哪几种对计算机系统进行分类的方法？各自反映的是计算机系统哪一个方面的特性？

11. 计算机可以应用在哪些领域？它对今天的社会发展和人们的生活方式有什么影响？

12. 下列选项中，描述浮点数操作速度指标的是________。

A. MIPS　　B. CPI　　C. IPC　　D. MFLOPS

13. 要学好本课程，应选择什么样的学习态度？需要注意哪些方面的问题？

第 2 章 数字电路基础和常用器件

本章首先介绍数字电路的基本元件，还给出了几个应用案例；包括数字电路基础及其处理方法；介绍基本逻辑关系、逻辑函数及其描述方法；在常用公式和基本规则的基础上，简要地介绍逻辑设计、逻辑函数化简。

其次介绍在计算机的逻辑部件中常用到的中小规模电路器件，包括组合逻辑电路和时序逻辑电路，这些是学习、设计、实现计算机硬件系统的必备知识。

最后介绍 CPLD 和 FPGA 等现场可编程逻辑器件，包括它们的内部组成和使用特性。

本章内容属于计算机组成原理课程的先修知识，读者可以根据自己的已有基础，采用浏览或者选学其中某些章节的办法来处理。

2.1 数字电路的基本元件

2.1.1 晶体二极管与三极管

1. 二极管

自然界中的物质根据其导电特性可以分为导体、绝缘体和半导体。锗、硅、砷化镓等由于它们的导电特性处于导体和绝缘体之间而被称为**半导体**。

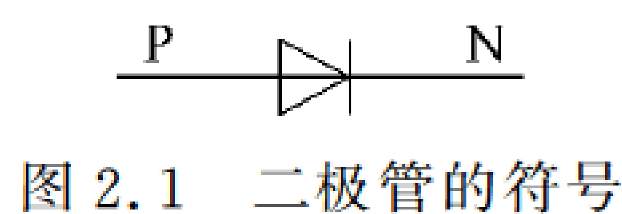


图 2.1 二极管的符号

在同一半导体基片上，分别制造 P 型半导体和 N 型半导体，经过载流子的扩散，在它们的交界面处就形成了 PN 结，PN 结加上管壳和引线，就成为半导体二极管(Diode)。二极管及其符号如图 2.1 所示。

当在 PN 结的某一个方向上加一个适当的电压后，这种器件可以导电，表现出一个导体的特性；而在相反的方向上加一个电压后，这种器件就几乎不导电，表现出一个绝缘体的特性，这就是二极管的单向导电特性。

为讨论问题方便，常常将晶体二极管看成一个理想的二极管，在数字电路中，理想二极管等效于一个开关，如图 2.2 所示。在加正向电压时，这个电子开关处于闭合状态，在加反向电压时，处于打开状态。

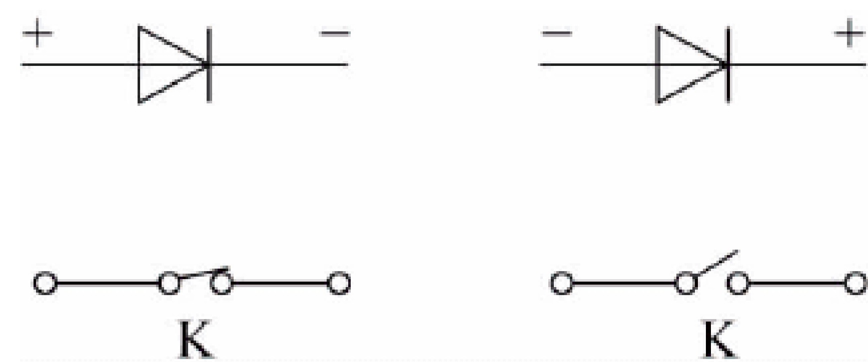


图 2.2 理想二极管等效于开关

2. 双极型三极管

双极型三极管也常称为晶体管(Transistor)，它可以通过控制基极电流来控制输出电流，其在数字电路中应用十分广泛。

它的结构由两个PN结(发射结、集电结)组成,包括NPN型和PNP型两种,3个引出电极是发射极e、基极b和集电极c,如图2.3所示。

双极型三极管的工作状态有饱和、截止和放大3种。数字电路中经常用前两种,即工作在开关状态,如图2.4(a)所示。理想的三极管也可以看成一个在基极b控制下的开关,即它在截止时打开,饱和时闭合。

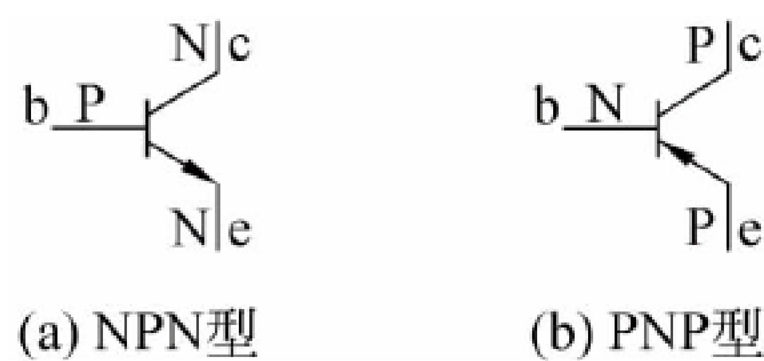


图2.3 双极型三极管

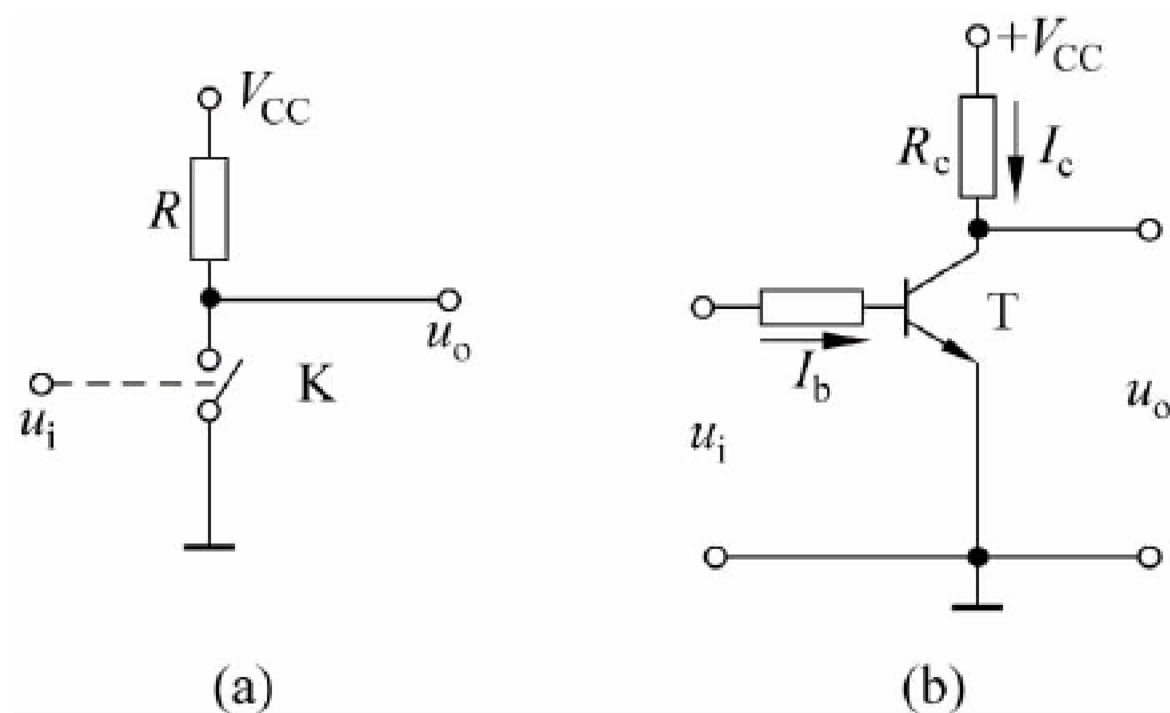


图2.4 反相器、可控开关

3. MOS管

金属-氧化物-半导体场效应三极管(Metal Oxide Semiconductor Field Effect Transistor,MOS管),是用电场控制输出电流的场效应半导体器件。其中增强型MOS管的结构与符号如图2.5所示。MOS管可分为N沟道(NMOS)和P沟道(PMOS)两种类型,任何一种都有3个与外部连接的电极,分别为源极(Source)、栅极(Gate)、漏极(Drain)。它的衬底一般与源极相连,衬底与栅极由绝缘层二氧化硅(SiO_2)隔开,大多数情况下,源极和漏极是可以互换的。

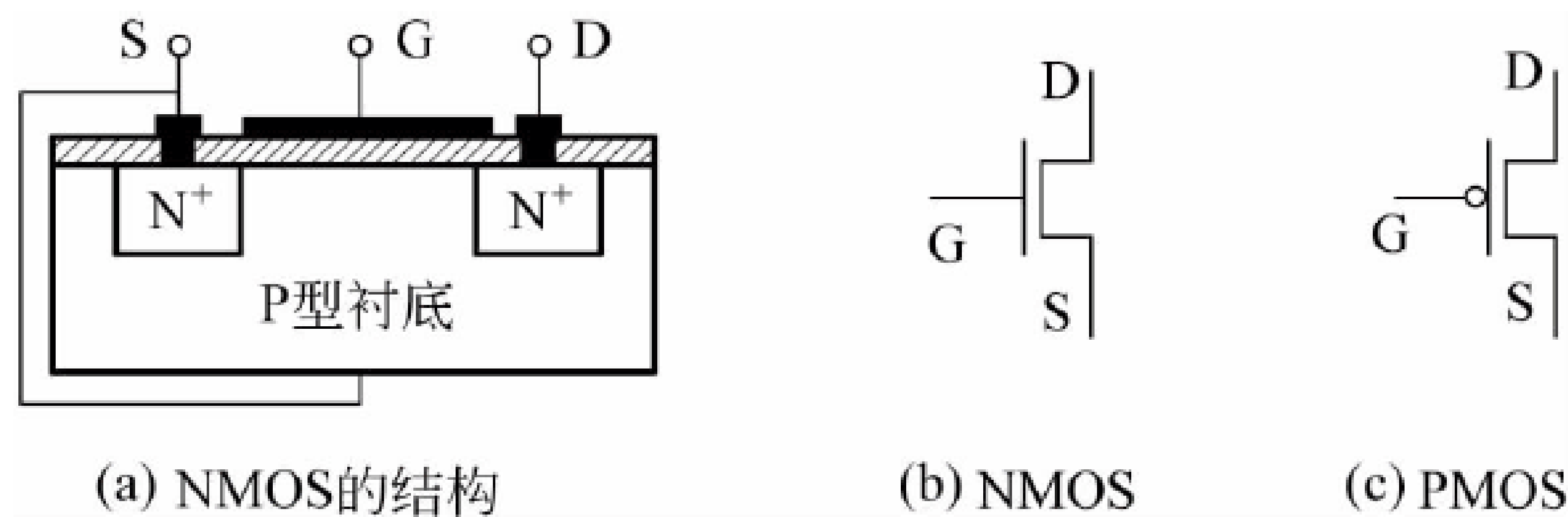


图2.5 增强型MOS管的结构及其符号

截止和深度饱和两种状态常常是MOS数字电路工作时所处的状态,图2.6(a)是一个NMOS反相器,理想的MOS管也是一个如图2.6(b)所示的可控开关。

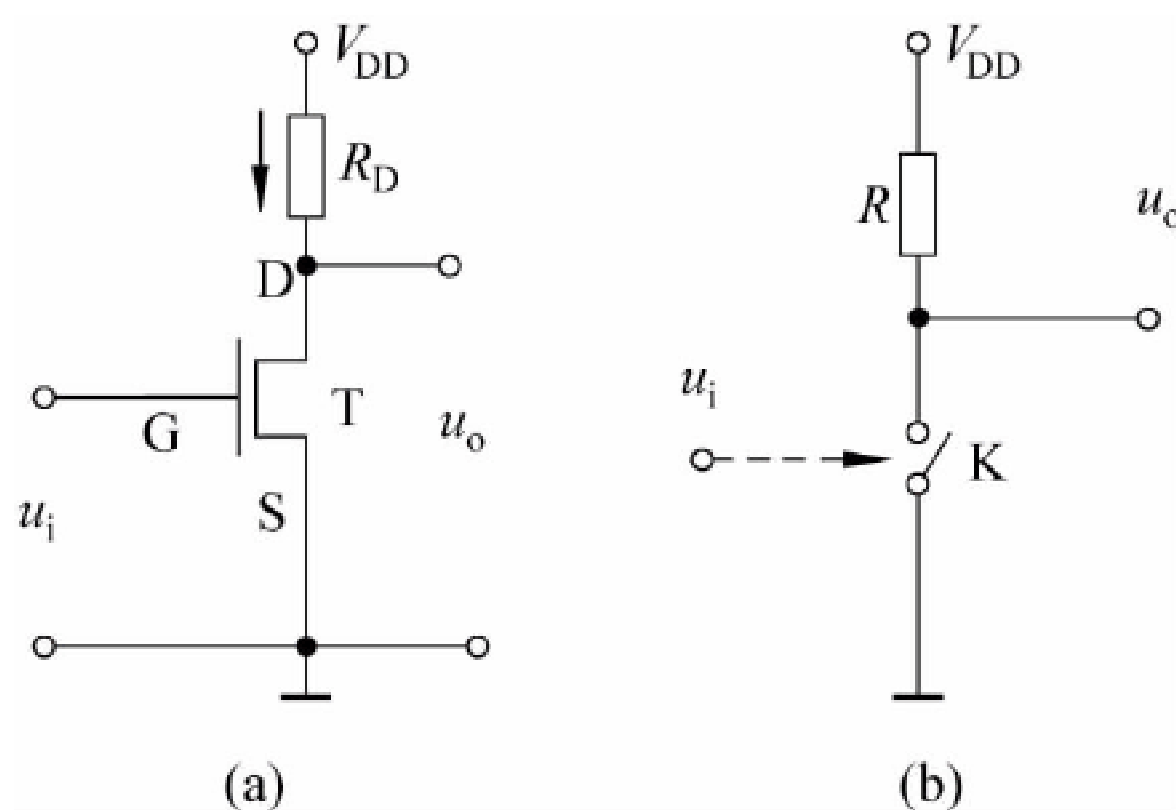


图2.6 NMOS反相器、理想MOS管

2.1.2 应用案例

1. 二极管应用案例

1）二极管门电路

二极管可以用作开关，构成各种门电路。图 2.7 是用二极管构成的一个与门电路。按理想二极管分析，D_1 管在正向电压的作用下，优先导通，相当于开关 K_1 合上，将 U_o 电位钳制在 0V，D_2 管则反偏截止，相当于开关打开，仅起隔离作用。

2）只读存储器

利用二极管的单向导电性，还可以构成只读存储器，如图 2.8 所示，输入的字线 X 和输出的位线 O 之间接一个二极管的位相当于存入 1 信号，无二极管的位相当于存入 0 信号。

2. 双极型三极管应用案例

利用三极管的开关特性，可以构成反相器、双稳态触发器电路和存储单元电路等。

三极管反相器如图 2.6 所示。

双极型三极管存储阵列元如图 2.9 所示，很容易用理想三极管的导通和截止特性来分析它的原理。

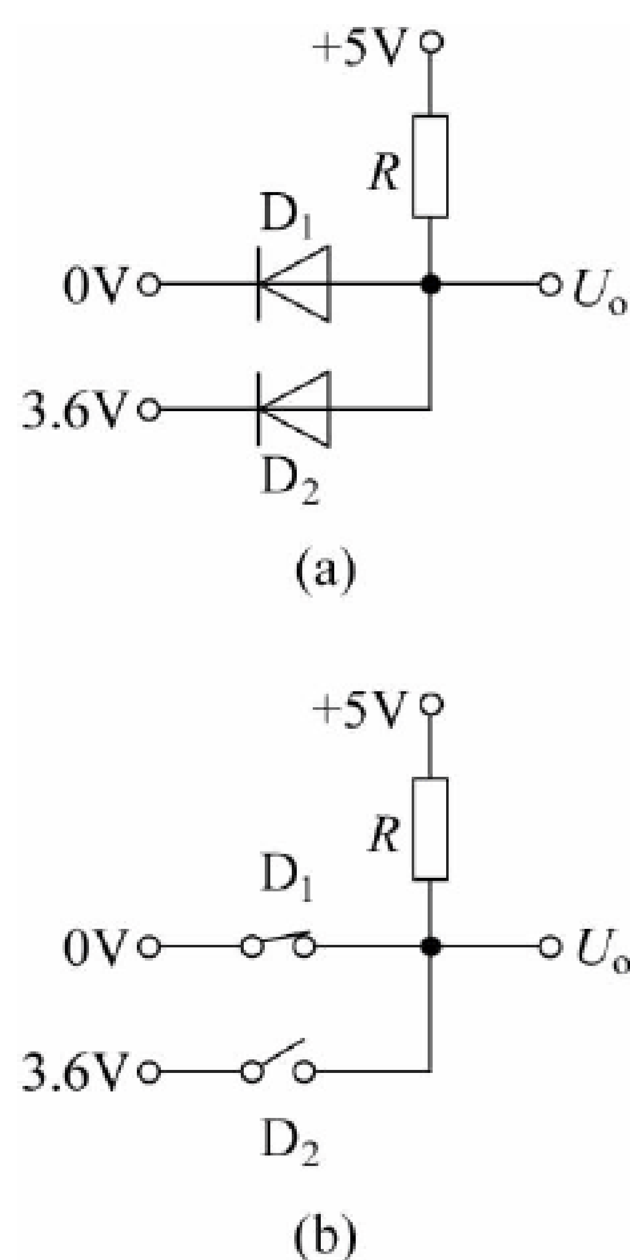


图 2.7 二极管门电路

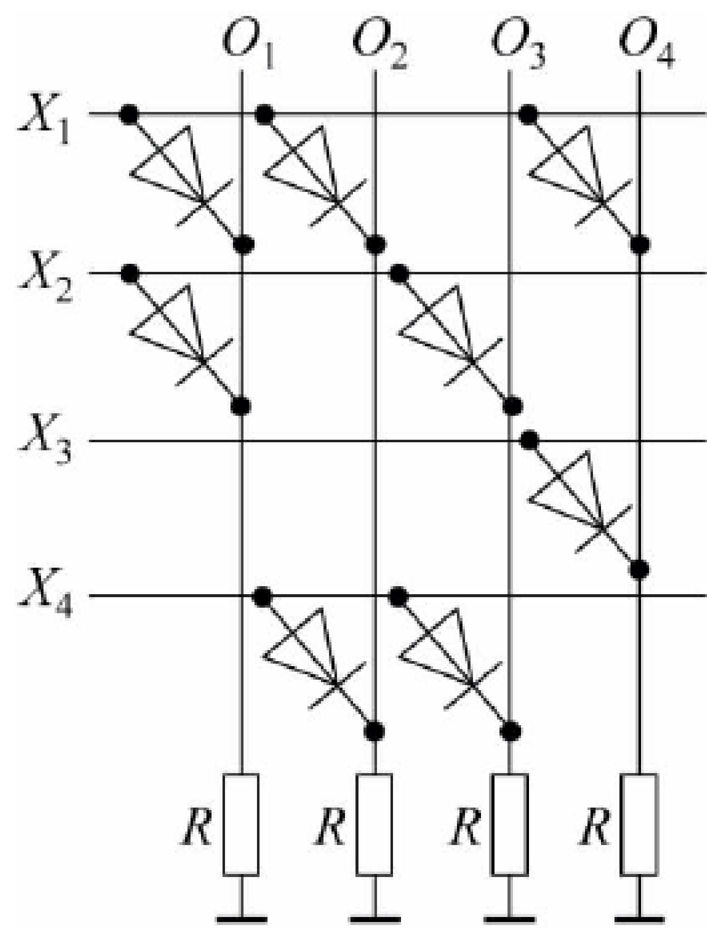


图 2.8 二极管只读存储器

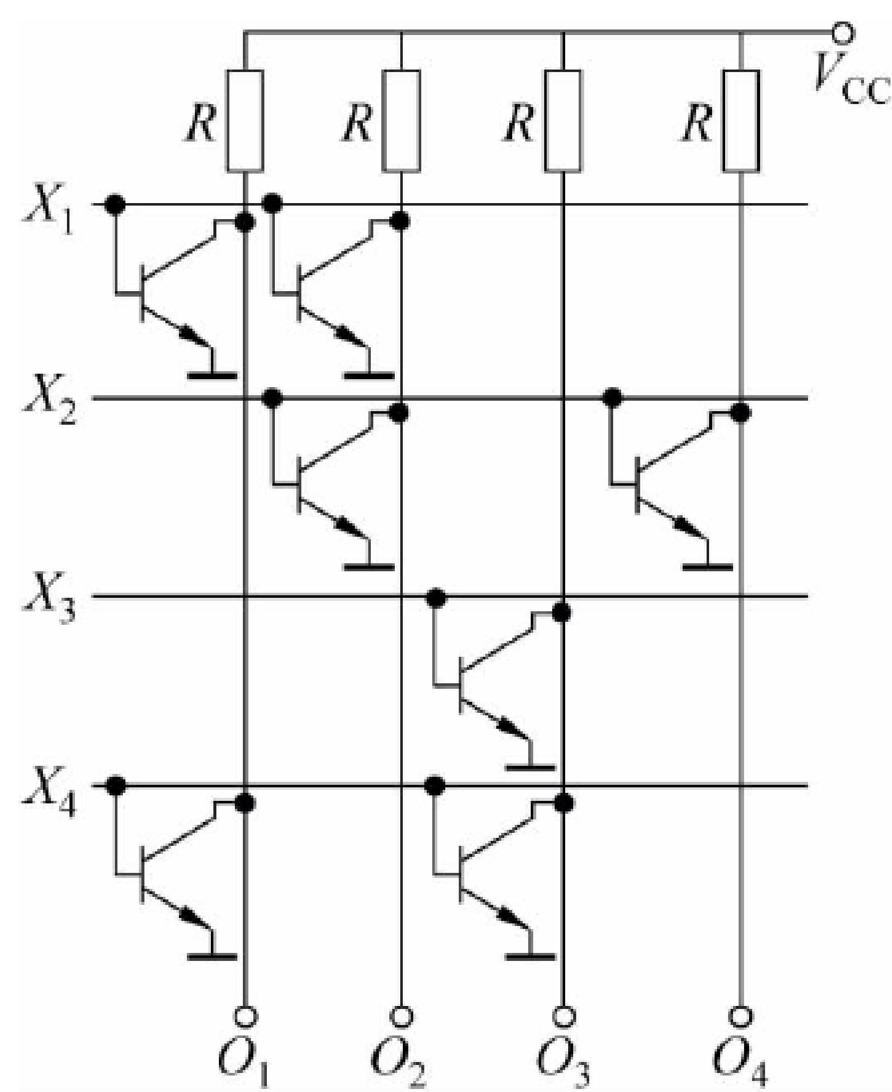


图 2.9 三极管只读存储器

3. MOS 管应用案例

利用 MOS 管的开关特性也可以构成许多基本的数字逻辑部件。

1）CMOS 反相器

CMOS 工艺是在电路中同时使用 NMOS 和 PMOS 管，图 2.10 是一个功耗极小的 CMOS 反相器。由理想 MOS 管可以容易地得出，当输入端为高电平（例如＋5V），NMOS 管导通，PMOS 管截止，输出为低电平（例如 0V）；反之输出为高电平（＋5V）。

2）单管动态存储器

图 2.11 是一个一位单管动态存储器基本电路。C 为存储 1 位信息的元件；T 为 NMOS 管；在字线为高电平时 T 导通（注意：大多数情况下，源极和漏极是可以互换的），这样才能

对该位存储器通过位线对它进行读写操作。

EPROM、EEPROM和闪存(Flash Memory)都是基于MOS管的工作原理。

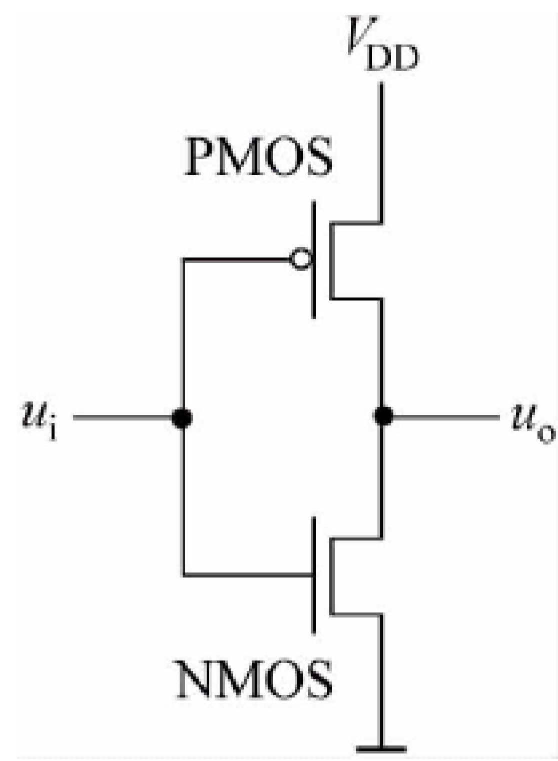


图2.10　CMOS反相器

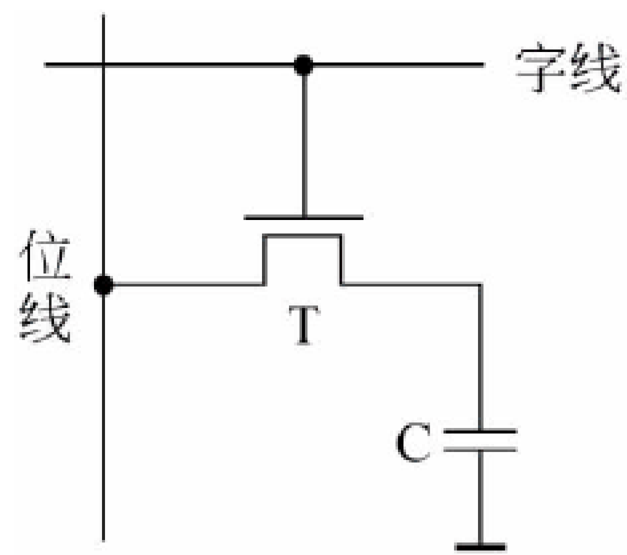


图2.11　一位单管动态存储器

2.2　数字电路基础及其相关处理方法

2.2.1　3种基本逻辑关系

数字电路通常由组合逻辑电路和时序电路组成。计算机硬件电路中的存储器、控制器、寄存器、译码器(地址、指令等)、加法器等,都是数字电路的具体应用,与、或和非门是构成这些部件最简单也是最基本的逻辑电路。

1. 与、或、非门

1) 与门

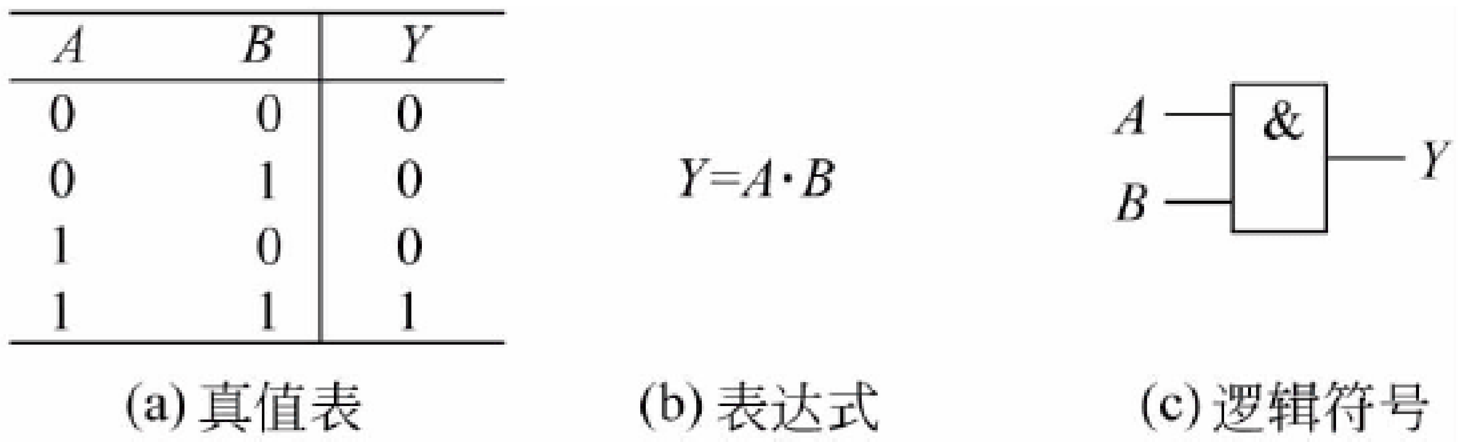
数字电路中,能实现如图2.12真值表所表示的逻辑关系:仅在全部的输入条件都具备(均为真)时,函数的输出才成立(为真),称这种关系为"与"逻辑,实现它的电路称为与门。真值表右边是与门电路的符号,表中的"1"和"0"代表的是日常生活中的"真"和"假"、"是"和"非"及"成立"和"不成立"等二值逻辑(Logic),并不表示数量的大小,而是表示两种对立的逻辑状态。

与逻辑的应用如某保险箱的开启规则:需要两位保管员 A 和 B 同时到场,否则不能开启。设保管员"到场"为"1",否则为"0";保险箱"开启"为"1",否则为"0"。将此关系填入如图2.12所示的一张表中,此表就是这个保险箱开启规则的真值表。

与门逻辑可以用如下表达式表示:

$$Y = A \cdot B$$

其中的"·"为逻辑"与"运算符号,表达式右边的 A 和 B 为自变量。



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

图2.12　与逻辑的真值表、门电路符号

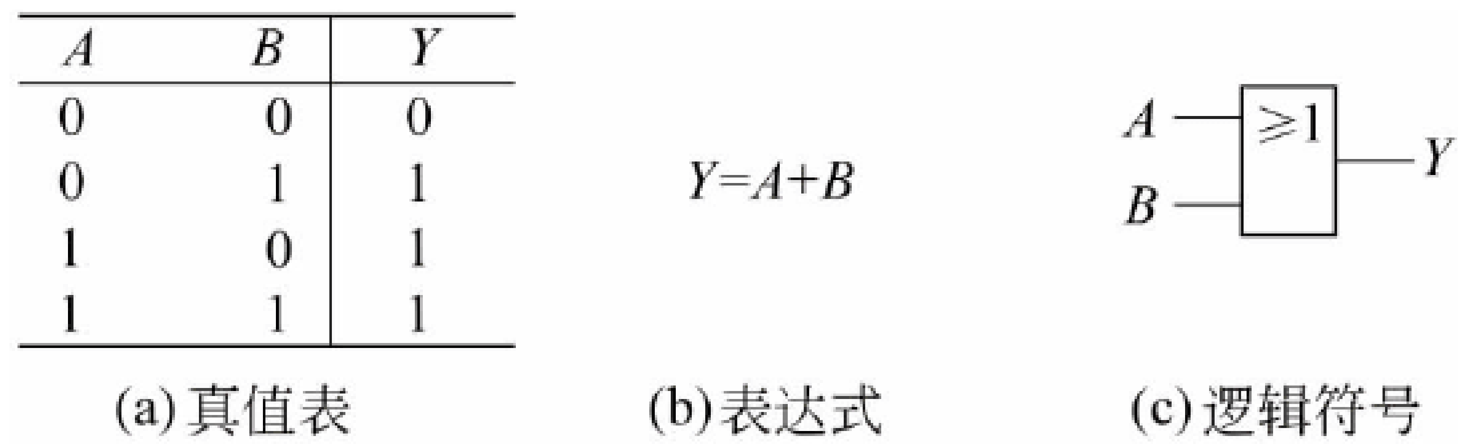
2）或门

如图 2.13 所示，能实现真值表所表示的关系：在全部的输入条件中只要有一个具备（为真）时，函数的输出就成立（为真），称这种关系为“或”逻辑。这种电路称为或门，真值表右边是或门电路的符号。

或门逻辑的表达式为

$$Y = A + B$$

其中的“＋”为逻辑“或”运算符号。



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

图 2.13　或逻辑的真值表、门电路符号

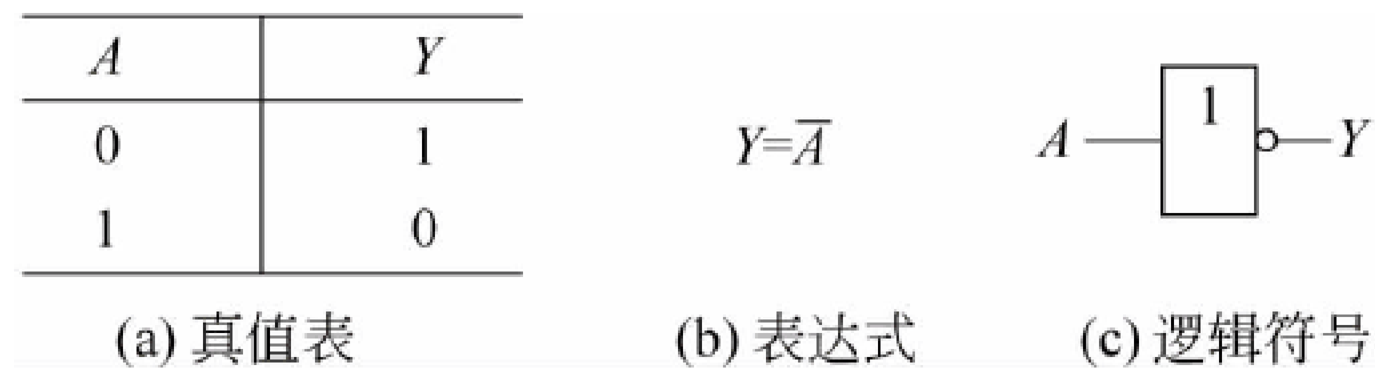
3）非门

如图 2.14 所示，能实现真值表所表示的逻辑求反的关系，称这种关系为“非”逻辑。这种电路称为非门，真值表右边是非门电路的符号。

非门逻辑的表达式为

$$Y = \overline{A}$$

其中的“—”为逻辑“非”的运算符号。



A	Y
0	1
1	0

图 2.14　非逻辑的真值表、门电路符号

选用这三种最基本的逻辑门（非门、与门、或门），就可以实现任何逻辑功能的电路，包括组合逻辑的电路和时序逻辑的电路。

2. 电路实现举例

如图 2.15 所示，二值逻辑的 0 和 1 在电路中以输出电平表示，通常用高电平表示 1，用低电平表示 0。注意，高低电平实际是一个范围，同样是高电平，但双极型的三极管和 MOS 管的电平范围有所不同。

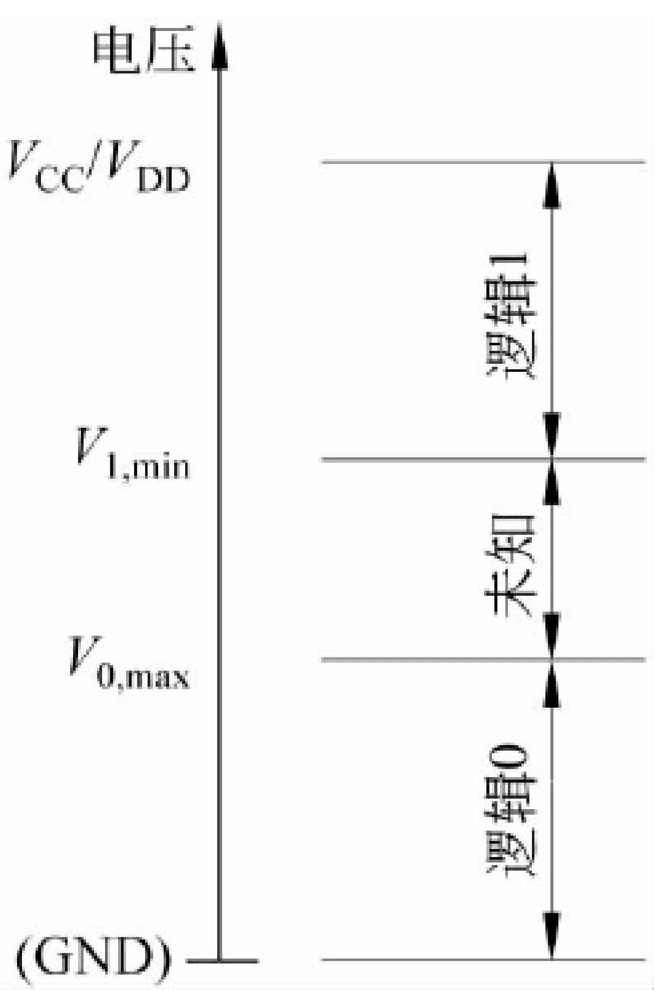


图 2.15　逻辑值的电平表示

如图 2.16 所示，为三极管实现的基本门电路的原理性电路。由理想三极管很容易地得出图 2.16（a）为非门，图 2.16（b）和图 2.16（c）分别为与门和或门。图 2.10 也是非门的电路实现，可见同一逻辑关系，可以由不同的电路来实现。

图 2.16 的门电路还存在一些如驱动能力、运行速度、可靠性等方面的问题，实际应用的门电路比这要复杂一些。CMOS 电路的与门的原理电路如图 2.17 所示。

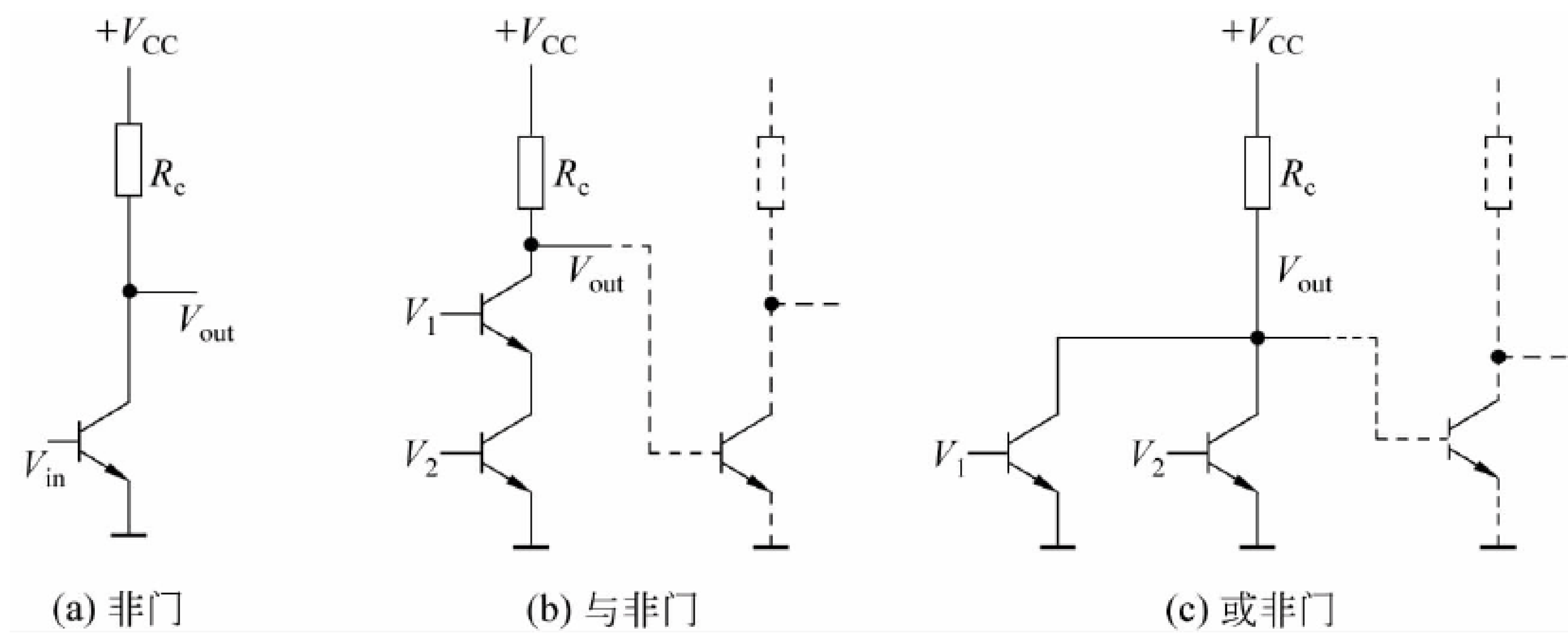


图 2.16　三种最基本门电路的原理性电路

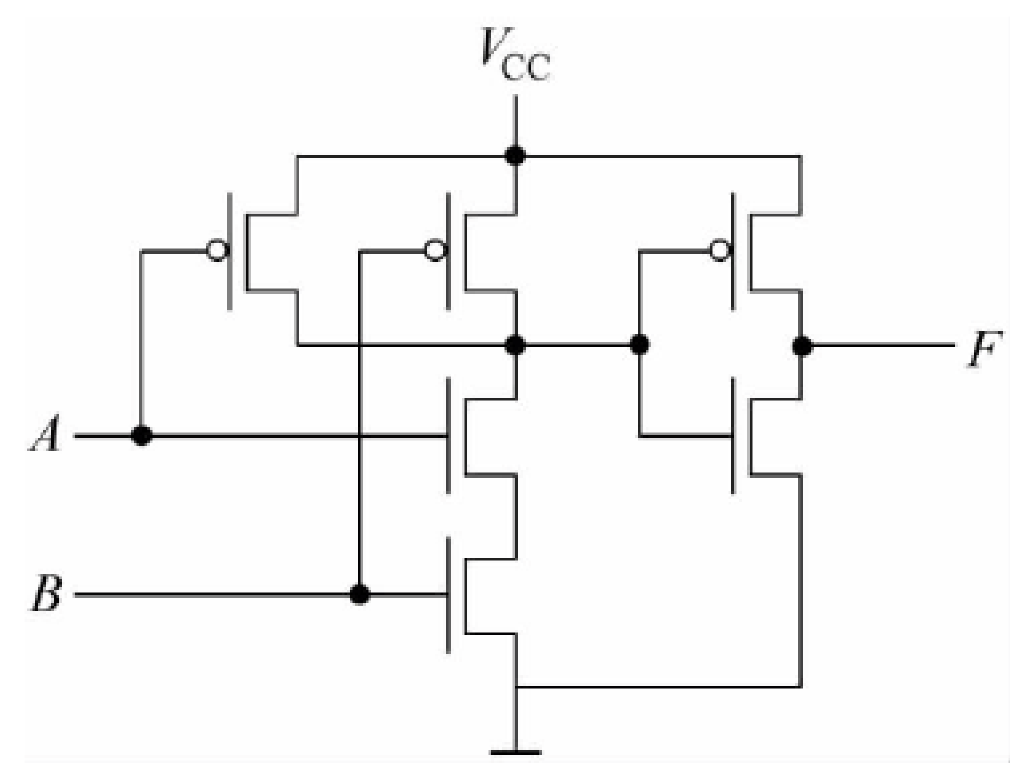


图 2.17　CMOS 电路的与门

3. 应用案例

1）与非门

图 2.18(a)是将一个与门和一个非门连接起来，构成常用的与非门。图右边是与非门的符号。同样，任何逻辑电路都可以由非和与非两种门表示。

2）或非门

图 2.18(b)是将一个或门和一个非门连接起来，构成常用的或非门。图右边是或非门的符号。任何逻辑电路都可以由非和或非两种门表示。

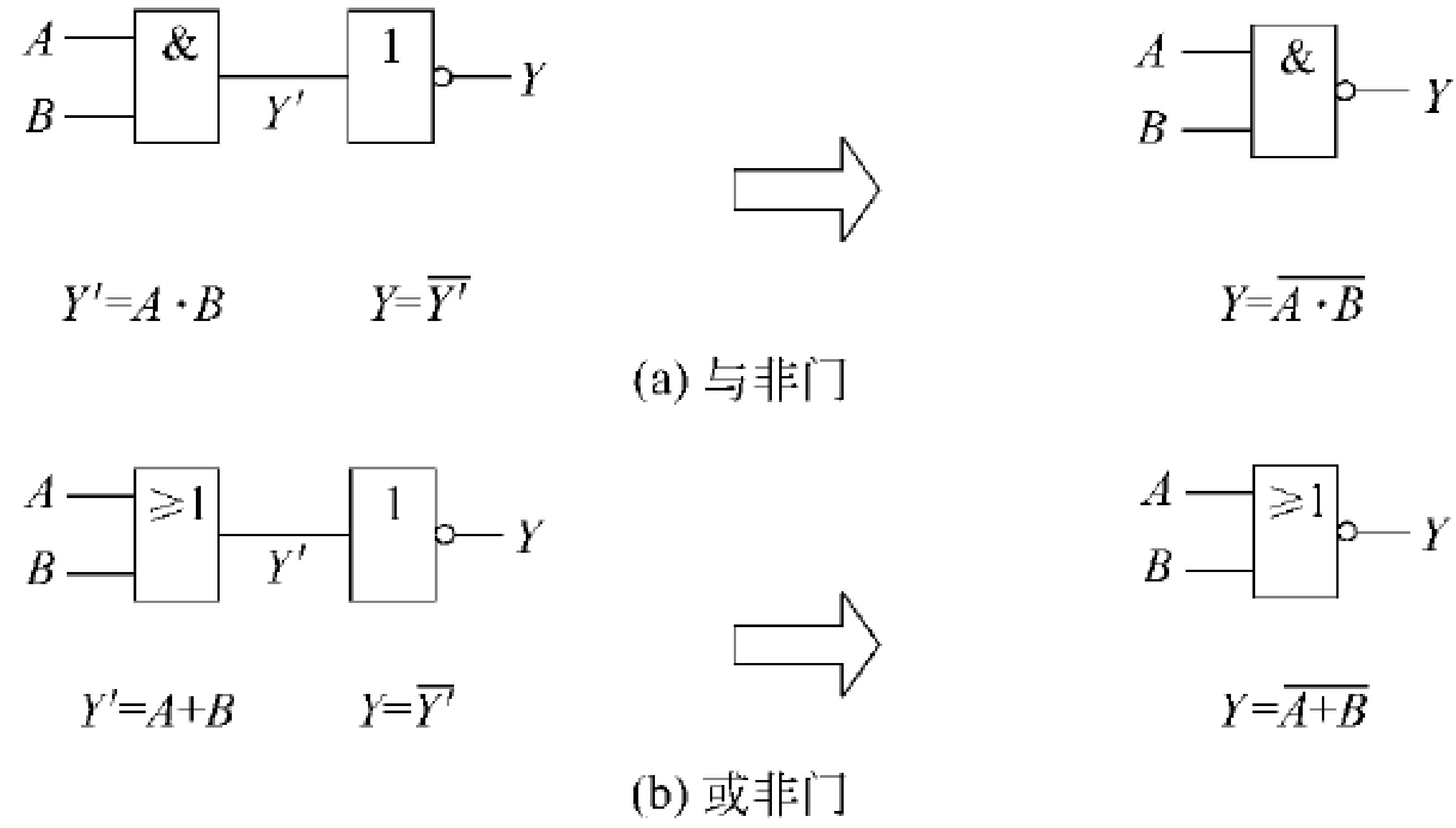


图 2.18　门电路应用案例

在计算机电路中，通常使用的基本逻辑门电路符号如图 2.19 所示，仅供参考。

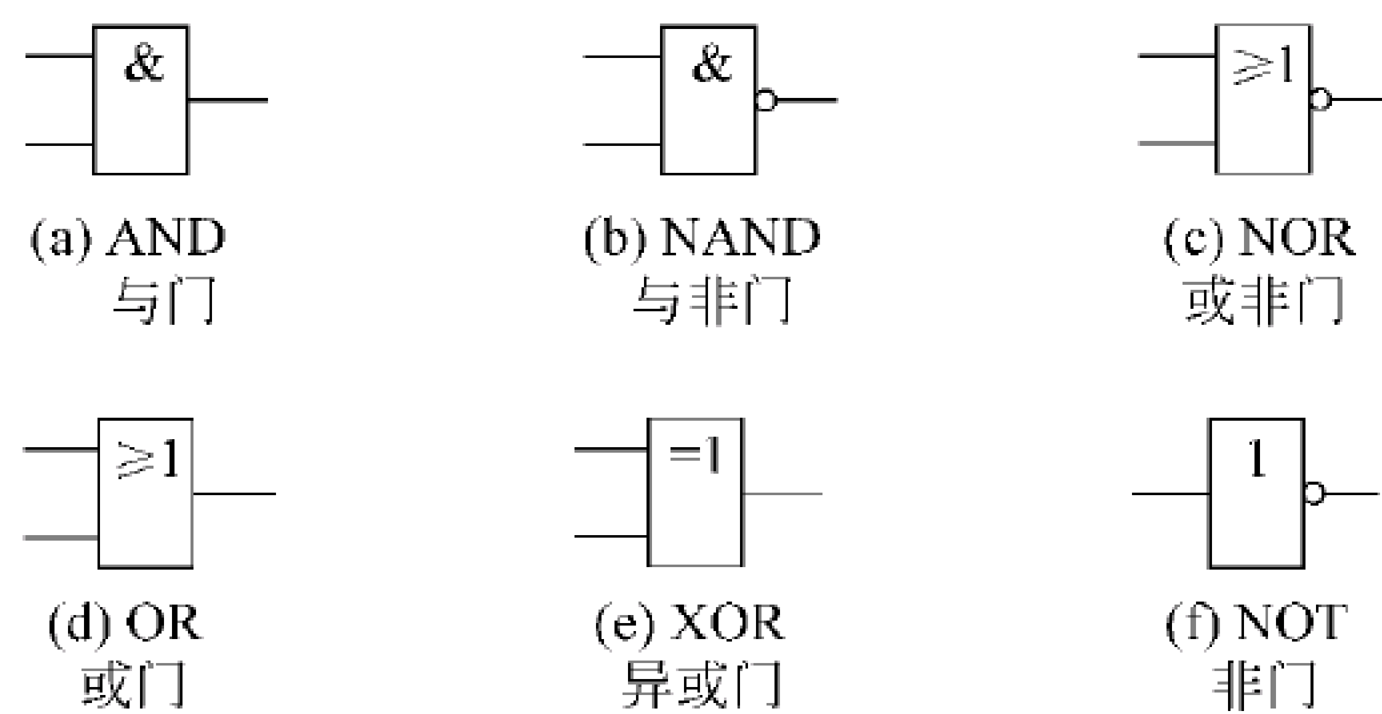


图 2.19　常用的逻辑门电路符号

2.2.2　逻辑函数及其描述方法

布尔代数研究二值逻辑(即逻辑代数),是分析和设计数字电路的数学工具,也是学习数字电路的基础。

1. 逻辑函数

布尔代数中输出变量与输入变量的逻辑关系称为逻辑函数,常写作

$$F = f(A,B,C\cdots)$$

逻辑函数通常采用表达式、真值表、逻辑图、波形图和卡诺图等方法表示。它们之间可以相互转换。

2. 逻辑函数的描述方法

1) 表达式

由逻辑变量与逻辑运算符构成的数学式称为逻辑表达式。例如:

$$F_n = X_n \cdot \overline{Y}_n + \overline{X}_n \cdot Y_n$$

也可以写成

$$F_n = X_n\overline{Y}_n + \overline{X}_nY_n$$

逻辑表达式运算的优先顺序依次为非、与、或。如想改变上式的运算顺序还要加括号。

2) 真值表

逻辑函数可以采用表格的形式表示,用于表示输入变量的各种可能取值与输出变量对应值的关系,这种表格称为真值表。表达式 $F_n=X_n\overline{Y}_n+\overline{X}_nY_n$对应的真值表如表 2.1 所示。

表 2.1　真值表

X_n	Y_n	F_n
0	0	0
0	1	1
1	0	1
1	1	0

一般由原始的逻辑关系可得到真值表。由真值表可写出相应的逻辑表达式,方法如下:

先将输出变量中取值为 1 的相应行中的输入变量取值相“与”;再将上述各与项“或”在一起。其中前者表示该行所满足的逻辑关系,后者表示完整的逻辑关系。

该方法通用于各种逻辑运算(简单或复杂的)关系。

用此方法很容易将表2.1的真值表转换成表达式 $F_n = X_n\overline{Y}_n + \overline{X}_nY_n$。

3) 逻辑图

逻辑函数可采用逻辑符号表示,这种形式的逻辑关系称为逻辑图。由表达式 $F_n = X_n\overline{Y}_n + \overline{X}_nYn$ 很容易地得到相应的逻辑图。图2.20就是用与、或、非门表示该表达式的逻辑图。这样的逻辑关系组成的逻辑电路也称为异或门,图左边是异或门的符号。异或关系在数字电路中也是常用的。

4) 波形图

逻辑函数可采用波形图表示,这种反映入出变量在时间上对应的逻辑关系称为波形图。表达式 $F_n = X_n\overline{Y}_n + \overline{X}_nY_n$ 随时间对应的波形图如图2.21所示。由波形图可以很直观地看出,F_n、X_n、Y_n 三者在 t_1 和 t_2 时刻均为两数相加之和的关系。这样的加法称为半加(不考虑低位来的进位和向高位的进位)。图2.20所示的电路也称为半加器。

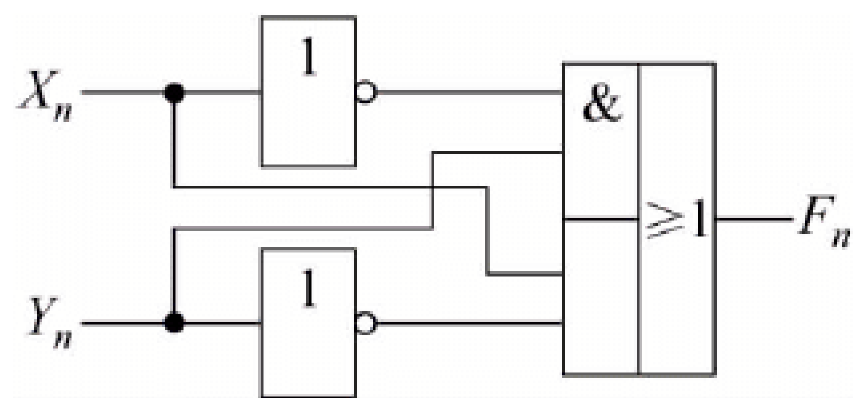


图2.20 半加器的逻辑图

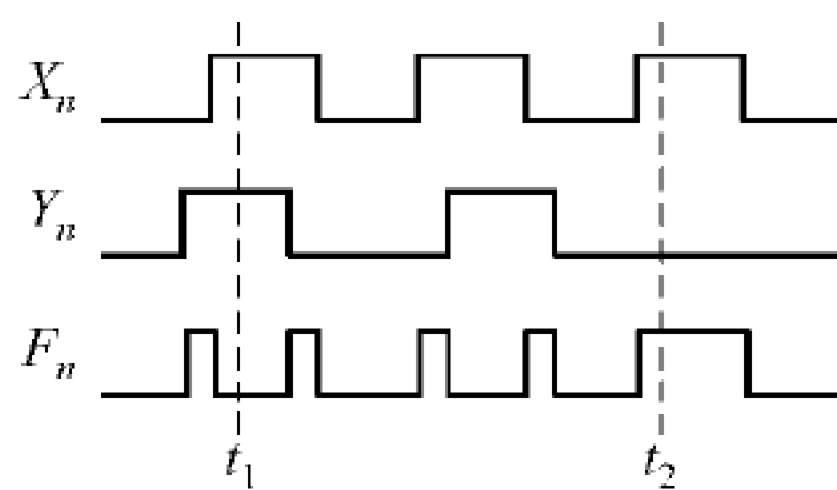


图2.21 半加器的波形图

2.2.3 逻辑函数的特性、规则与应用

通常,某个逻辑函数的表达式比较简单,则实现它所需要的电路元件就少,这既节约了器材、提高了电路的可靠性,又有利于缩短信号的传输延迟时间。因此,逻辑表达式的化简对设计逻辑电路是必不可少的。

1. 布尔代数的基本特性

$A+1=1 \qquad A\cdot 0=0$

$A+0=A \qquad A\cdot 1=A$

$\overline{\overline{A}}=A$

$A+A=A \qquad A\cdot A=A$

$A+\overline{A}=1 \qquad A\cdot\overline{A}=0$

交换律:$A+B=B+A \qquad A\cdot B=B\cdot A$

结合律:$(A+B)+C=A+(B+C)$

$(AB)C=A(BC)$

分配律:$A(B+C)=AB+AC$

反演律(德·摩根律):$\overline{AB}=\overline{A}+\overline{B} \qquad \overline{A+B}=\overline{A}\,\overline{B}$

吸收律:$A+AB=A \qquad A(\overline{A}+B)=AB$

$A+\overline{A}B=A+B \qquad (A+B)(A+C)=A+BC$

2. 三个重要的规则

1) 代入规则

用一个逻辑函数替代逻辑等式两边所有的某一变量，则等式仍然成立。

例：化简 $F=XYZ+X(\overline{YZ})$，设 $H=YZ$，根据代入规则有

$$F = XH + X\overline{H} = X$$

2）反演规则

对函数 F 中所有的变量取反、与变或、或变与、0 变 1、1 与 0，即得到其反函数 $\overline{F}$。

例：$F_n=X_n \cdot \overline{Y}_n+\overline{X}_n \cdot Y_n$，根据反演规则可得到它的反函数为

$$\overline{F}_n = (\overline{X}_n + Y_n) \cdot (X_n + \overline{Y}_n)$$

注意：*在使用反演规则时，原有运算顺序必须保持不变，必要时可以加括号。*

原函数 $F_n=X_n \cdot \overline{Y}_n+\overline{X}_n \cdot Y_n$ 的表达式中，有 $(X_n \cdot \overline{Y}_n)$ 和 $(\overline{X}_n \cdot Y_n)$ 两个与项，然后再将两个与项相或，称这种由若干个与项相或形式组成的表达式为与或式，这是逻辑代数中最常见的逻辑函数表达式。

3）对偶规则

将函数 F 中与变为或、或变为与、0 变为 1、1 变为 0，即得到一个新的函数 F'，并称 F' 为 F 的对偶式。当某一等式成立，则它的对偶式也成立。

例：$F_1=f_1(A,B,C)$，$F_2=A+\overline{B}C$，且 $F_1=F_2$；现有 $F_1{}'=f_1{}'(A,B,C)$ 为 F_1 的对偶式，$F_2{}'=A(\overline{B}+C)$，证明 $F_1{}'=F_2{}'$。

因为 $A(\overline{B}+C)$ 是 $A+\overline{B}C$ 的对偶式，所以 $F_2{}'$ 为 F_2 的对偶式，且已知 $F_1{}'$ 为 F_1 的对偶式。

又因 $F_1=F_2$，由对偶规则，所以 $F_1{}'=F_2{}'$ 成立。

3. 与或式的化简

逻辑函数的化简有公式化简、卡诺图(Karnaugh Map)化简（图解化简）和表格化简（Q-M 化简）3 种方法。由于通常情况下通过工程软件可以优化（自动化简），所以作为原理此处仅讨论公式化简。其他化简方法可参考有关书籍的相关内容。

通常一个逻辑函数所对应的逻辑表达式不唯一，因而这些表达式的繁简程度也各异。一般对于不同类型的表达式而言，“简单”的标准也各不相同。就常见的与或表达式而言，“最简”的含义指该表达式与项个数最少，并且各与项所含的变量个数也最少。

运用逻辑代数的特性对表达式进行化简称为公式化简。

例：运用 $A+\overline{A}=1$，可以将两项**合并**为一项，从而消去一个变量，使

$$A\overline{B}C + A\overline{B}\overline{C} = A\overline{B}(C+\overline{C}) = A\overline{B}$$

例：运用 $A+\overline{A}B=A+B$，可以**消去**多余的因子，使

$$\overline{A} + AB + DE = \overline{A} + B + DE$$

例：运用 $A+AB=A$，可以**吸收**多余的项，使

$$A\overline{B} + A\overline{B}CD(E+F) = A\overline{B}$$

2.3 组合逻辑电路及时序逻辑电路

数字电路通常由组合逻辑电路（Combinational Logic Circuit）和时序逻辑电路（Sequential Logic Circuit）组成。组合逻辑电路中，任一时刻电路的输出状态仅与当前输入信号的状态有关，与电路原来的输出状态无关，没有记忆功能。其电路结构为信号从输入端

逐级向输出端传输,没有后级向前级的反馈。

本节给出常用的实现组合逻辑功能的中小规模集成电路,包括它们的器件功能、引脚分配等内容。

2.3.1 常用逻辑门器件

1. 基本逻辑门

在教学计算机中选用的基本逻辑门有非门、与(非)门等,型号为6反相器SN74LS04、4-2输入正与非门SN74LS00、4-2输入正与门SN74LS08,如图2.22所示。用于实现信号反相(变极性)或最简单的逻辑处理功能。

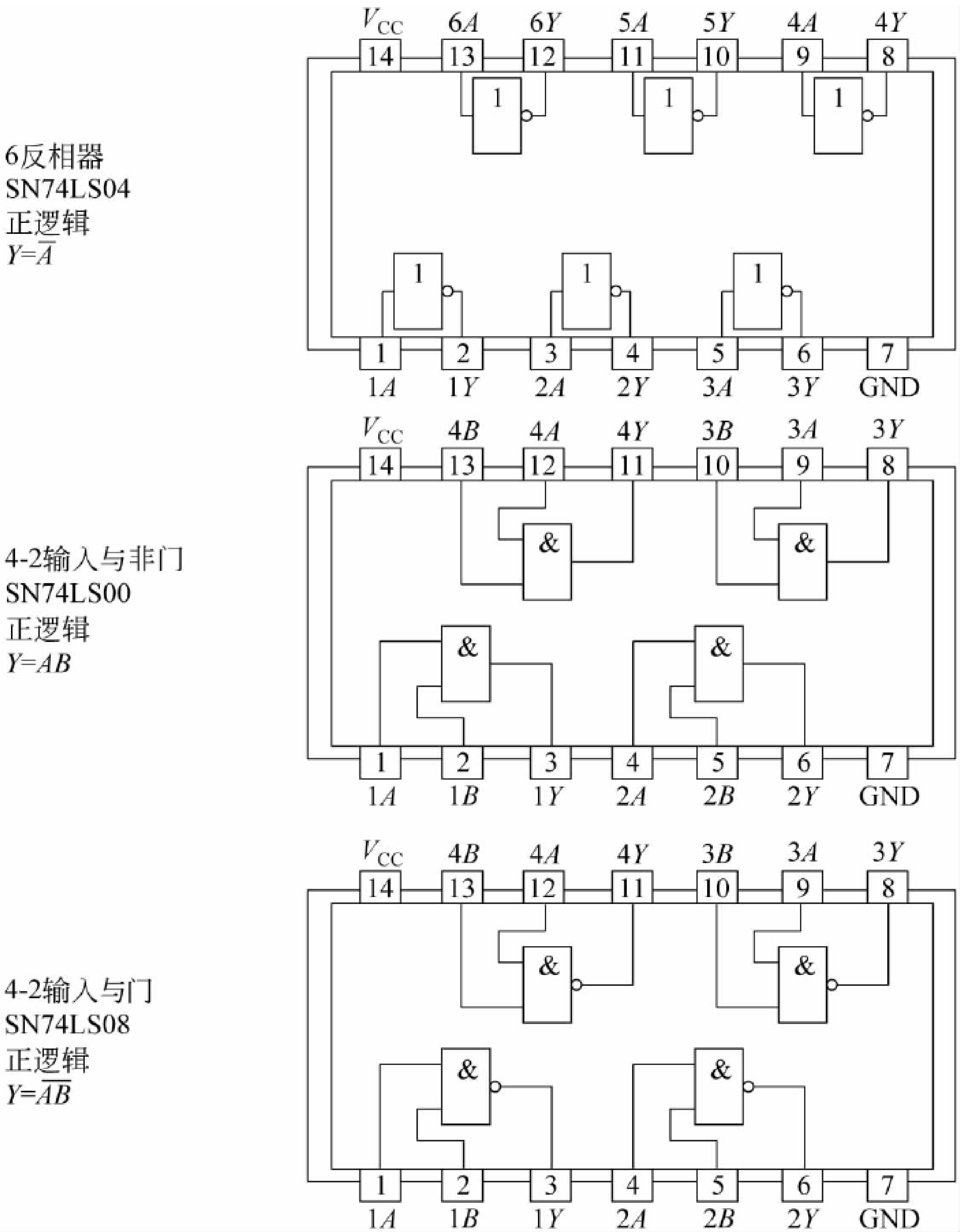
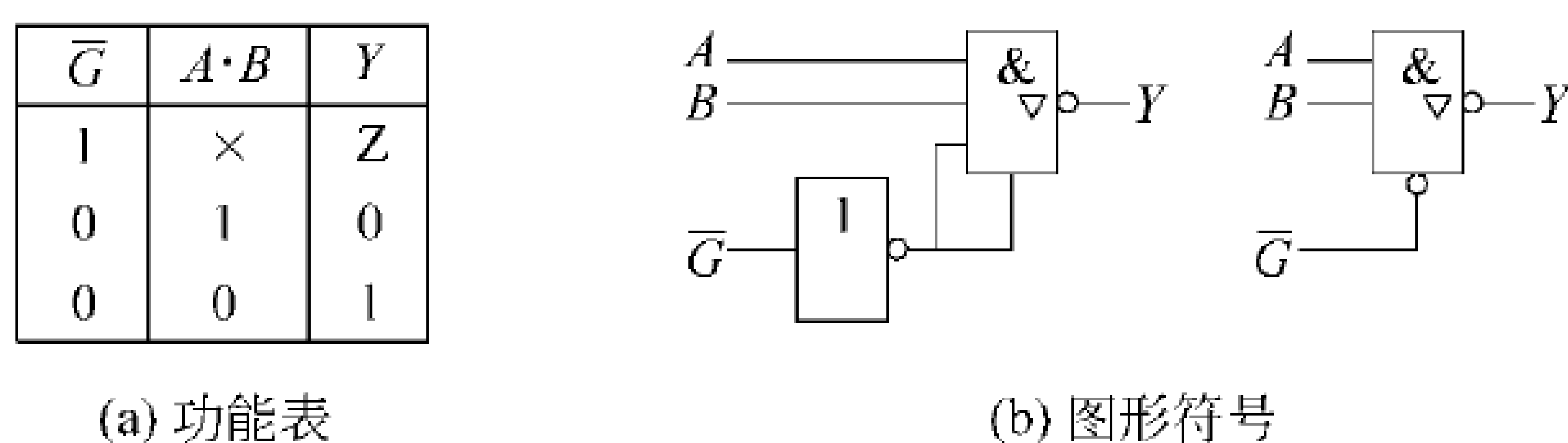


图2.22 基本逻辑门电路

2. 三态门器件

三态门是具有3个输出状态的逻辑电路,比二值逻辑电路多一个高阻状态Z,是常用的总线接口电路。其功能和电路符号如图2.23所示。

三态电路的输出状态是通过一个输入$\overline{G}$控制的。当$\overline{G}$为低(有效)时,三态电路的输出取决于输入状态,给出正常的0或1输出;当$\overline{G}$为1时,输出为高阻态。图2.23(a)中的×



$\overline{G}$	$A\cdot B$	Y
1	×	Z
0	1	0
0	0	1

(a) 功能表

图 2.23　三态与非门的功能表与符号

表示随意态，其值为 0 或为 1 均可。

三态电路最重要的应用是构成硬件中总线的接收器和发送器。常用的有实现单向传送功能的 SN74LS240 和 SN74LS244，二者的数据线引脚分配相同，但前者入出的极性相反，后者入出的极性相同，两部分的 4 位数据分别由 $2\overline{G}$ 和 $1\overline{G}$ 控制输出是否高阻态。还有实现双向传送功能的 SN74LS245，分别由 $\overline{G}$ 和 DIR 控制 A 与 B 是否连通以及信息传送方向。器件的内部逻辑关系和外部数据线引脚如图 2.24 所示。

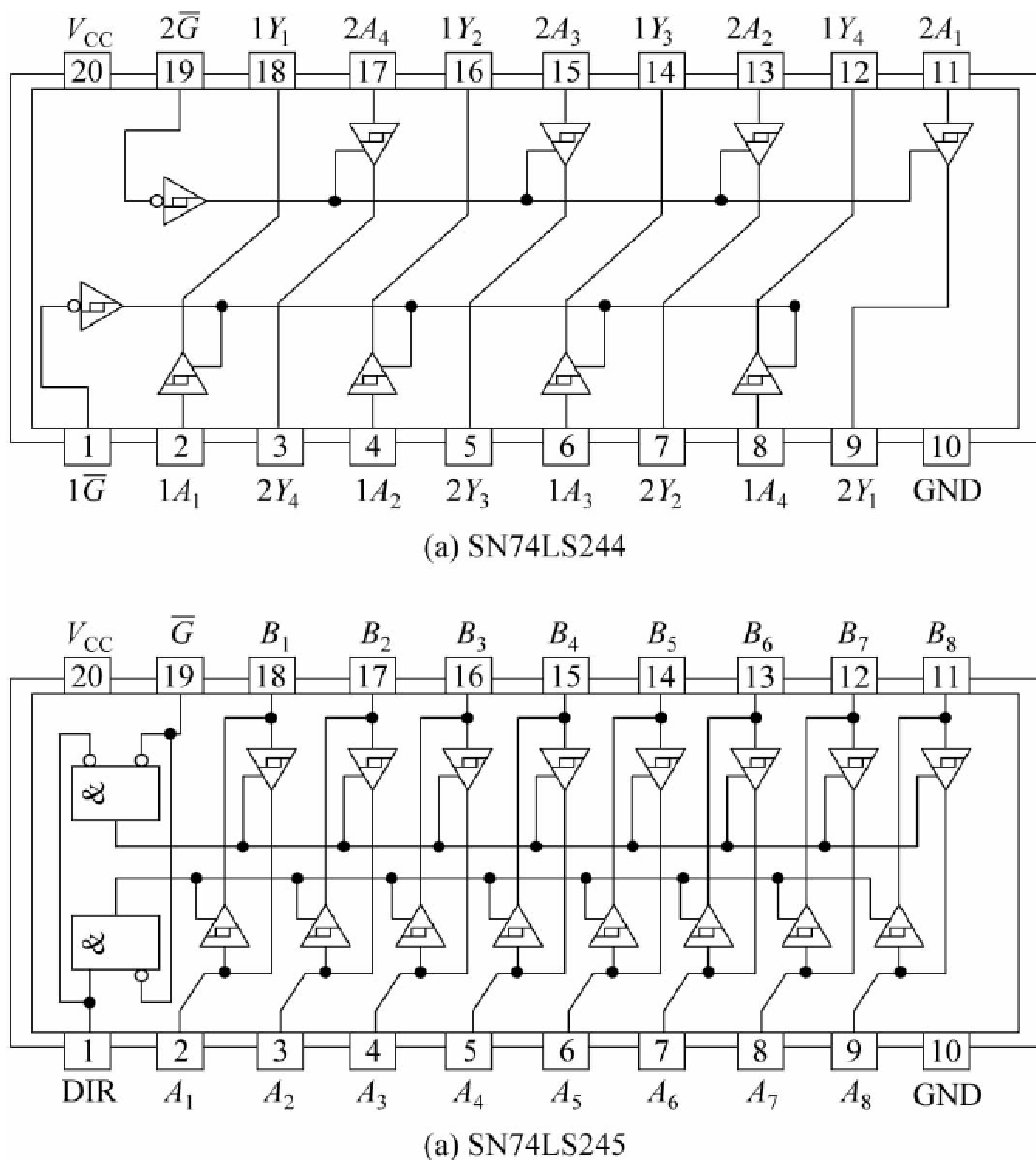
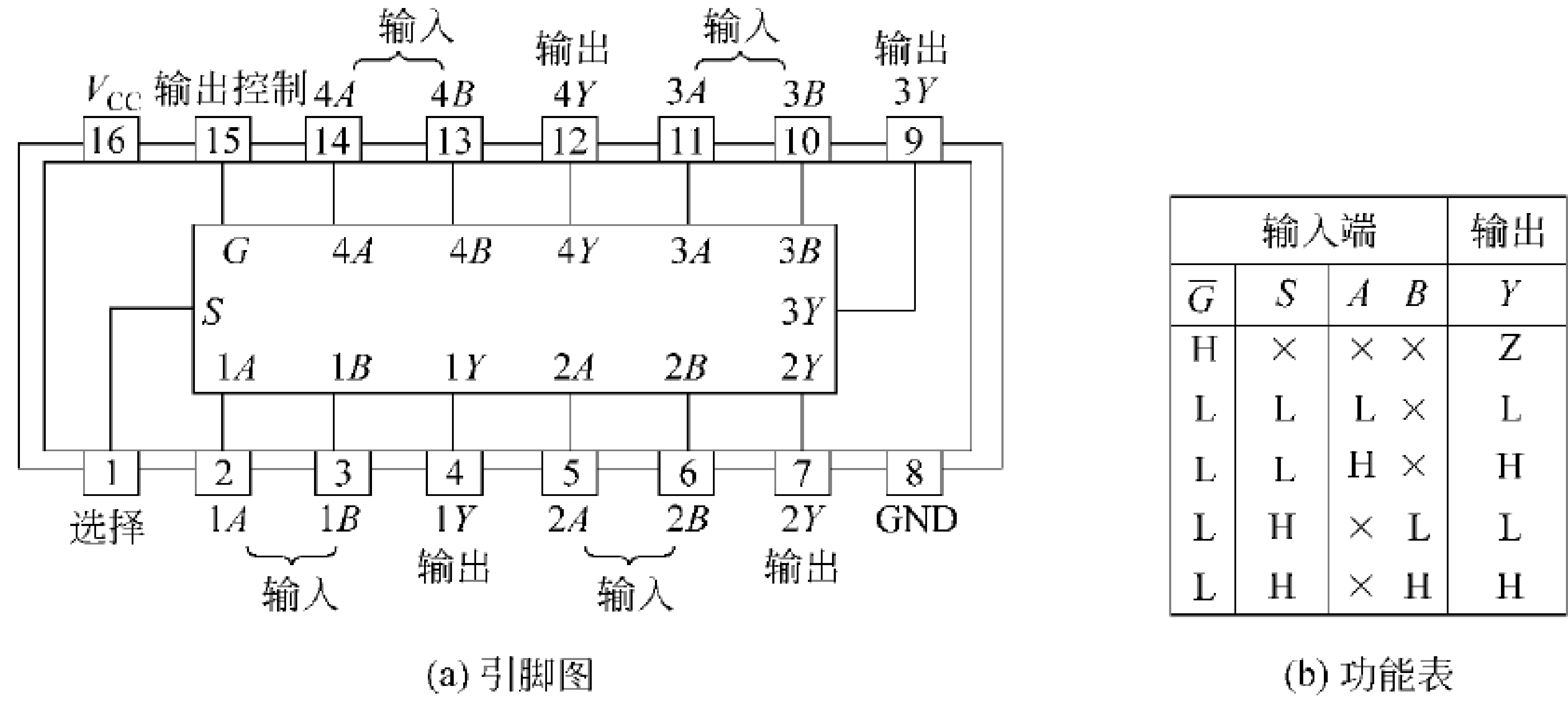


图 2.24　SN74LS244 和 SN74LS245 的引脚图

3. 多路选择器

多路选择器又称多路开关，实现从多路的输入数据中选择一路作为输出的功能。教学计算机中选用了 4 位带有三态输出控制的 SN74LS257 器件，如图 2.25 所示。

输入选择控制信号 S，用于选择将 A 路输入还是 B 路输入送到输出。当 S 为低时，输出来自 A 路输入，反之来自 B 路输入。输出控制信号 $\overline{G}$ 为低，被选中的 4 位输出为正常逻辑电平，反之则输出高阻态。



(a) 引脚图

输入端				输出
$\overline{G}$	S	A	B	Y
H	×	×	×	Z
L	L	L	×	L
L	L	H	×	H
L	H	×	L	L
L	H	×	H	H

(b) 功能表

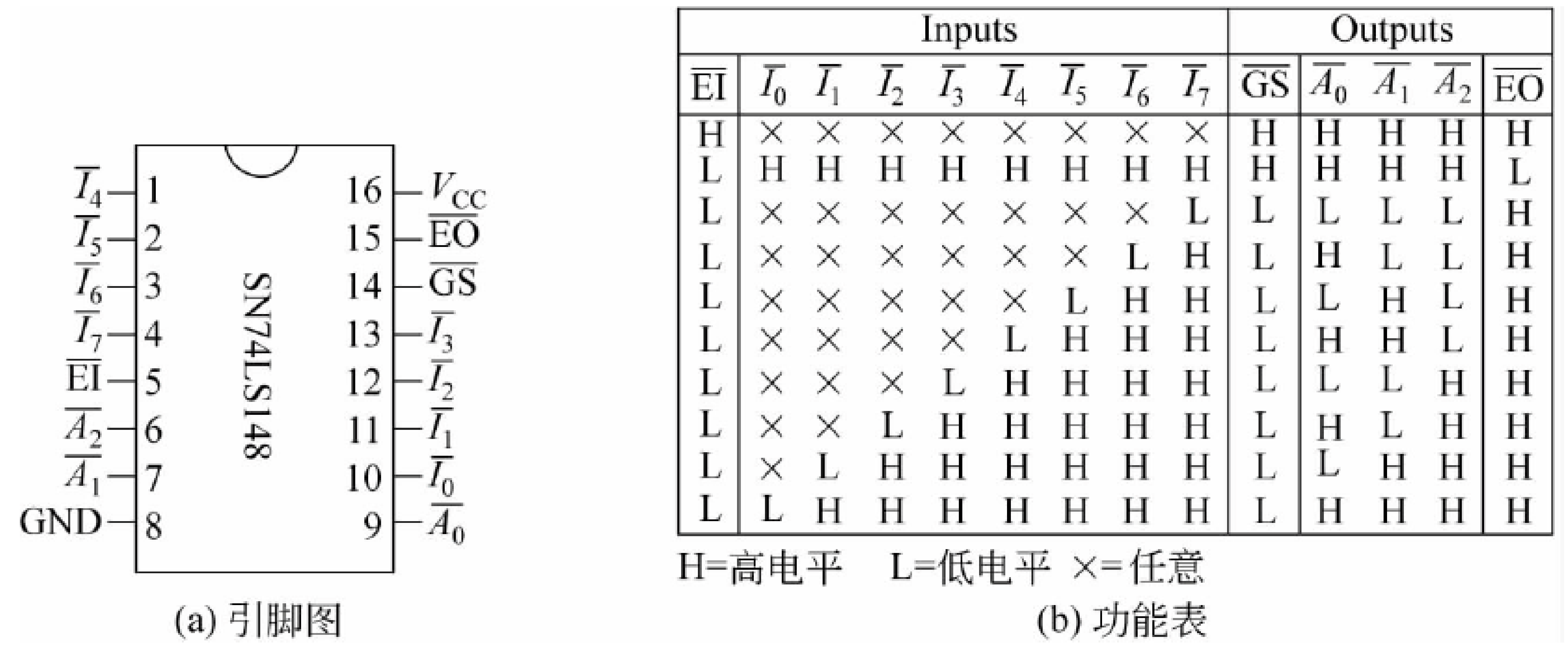
图 2.25　SN74LS257 引脚图与功能表

4. 编码器和译码器

计算机中的信息大都是被编码的。编码器是把输入信号转换成需要的编码。一般是把 2^n 个输入信号不同的状态组合,按预先规定的优先级,编码成 n 位输出信号。编码器的种类很多,计算机中常用到优先编码器,例如 SN74LS148 优先编码器。该编码器有 8 位输入、3 位输出,功能表如图 2.26 所示。

$\overline{\text{EI}}$(低有效)为编码控制信号,$\overline{\text{EO}}$、$\overline{\text{GS}}$为输出状态信号,用于实现多个同类器件之间的级联。仅当$\overline{\text{EI}}$为低,编码器才进行识别与编码。当 8 位输入从右向左遇到第 1 个低,另 7 位可能为高或低时,编码器才输出正常的 3 位编码(对应输入从右向左第 1 个低优先),此时$\overline{\text{GS}}$和$\overline{\text{EO}}$为 01,表明编码有效。

若 8 位输入均为高,或$\overline{\text{EI}}$为高,编码器 3 位输出均为高。并通过$\overline{\text{GS}}$和$\overline{\text{EO}}$为 10 或 11 来表明两种不同情形。



(a) 引脚图

Inputs									Outputs				
$\overline{\text{EI}}$	$\overline{I_0}$	$\overline{I_1}$	$\overline{I_2}$	$\overline{I_3}$	$\overline{I_4}$	$\overline{I_5}$	$\overline{I_6}$	$\overline{I_7}$	$\overline{\text{GS}}$	$\overline{A_0}$	$\overline{A_1}$	$\overline{A_2}$	$\overline{\text{EO}}$
H	×	×	×	×	×	×	×	×	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	×	×	×	×	×	×	×	L	L	L	L	L	H
L	×	×	×	×	×	×	L	H	L	H	L	L	H
L	×	×	×	×	×	L	H	H	L	L	H	L	H
L	×	×	×	×	L	H	H	H	L	H	H	L	H
L	×	×	×	L	H	H	H	H	L	L	L	H	H
L	×	×	L	H	H	H	H	H	L	H	L	H	H
L	×	L	H	H	H	H	H	H	L	L	H	H	H
L	L	H	H	H	H	H	H	H	L	H	H	H	H

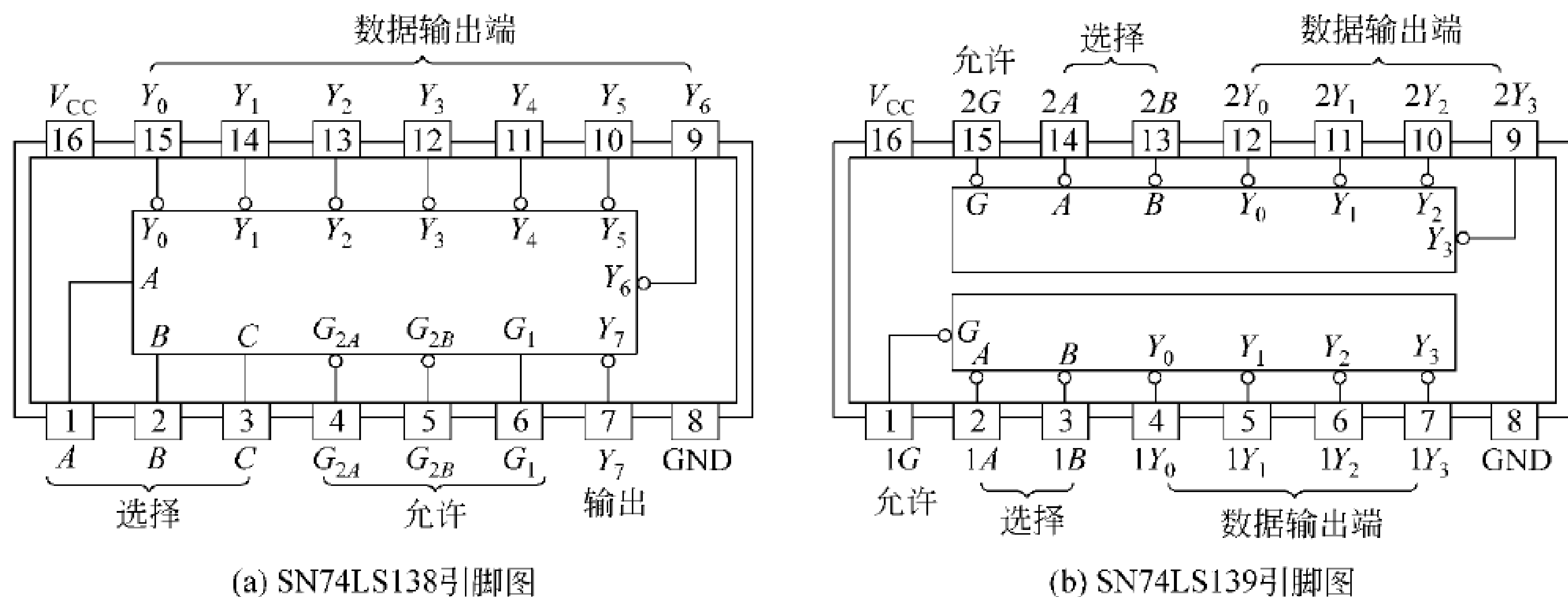
H=高电平　L=低电平　×=任意

(b) 功能表

图 2.26　SN74LS148 引脚图与功能表

译码是编码的逆过程,即将输入的编码转换成原来的信号表示。与编码器一样,译码器的种类也很多,计算机中常用于把一组输入代码的状态组合翻译成相应的控制电位。若输入信号有 n 个,其输出最多可以有 2^n 个。正常输出时,2^n 个输出中仅有一个输出为低(或高),其他 2^n-1 个输出均为高(或低),用于表明 n 个输入的某一种状态组合;不

需要译码时，通过另外的控制信号可以使输出全部为高(低)，用于表明不选择任何输入状态组合。教学计算机中，选用了有 3 个输入信号、8 个(2^3)输出信号(低有效)的 SN74LS138 译码器(常称 3-8 译码器)，也选用了双路的各有 2 个输入信号、4 个(2^2)输出信号(低有效)的双 2-4 译码器 SN74LS139。它们的引脚、内部逻辑与功能表如图 2.27 所示。



(a) SN74LS138引脚图

(b) SN74LS139引脚图

输入端					输出端							
允许		选择										
G_1	G_2^*	C	B	A	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
×	H	×	×	×	H	H	H	H	H	H	H	H
L	×	×	×	×	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

*$G_2=G_{2A}+G_{2B}$

(c) SN74LS138功能表

输入端			输出端			
允许	选择					
G	B	A	Y_0	Y_1	Y_2	Y_3
H	×	×	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

(d) SN74LS139功能表

图 2.27 译码器 SN74LS138 和 SN74LS139 的引脚图与功能表

2.3.2 时序逻辑电路

1. 基本 R-S 触发器和锁存器

计算机系统能够自动并协调一致地工作，是在控制器控制下严格地按时序工作，时序电路是计算机电路中的重要组成部分。它的特点是任一时刻，电路的输出，不仅与该时刻的输入有关，还与电路以前的状态有关(即有记忆功能)。

教学计算机中使用了 R-S 触发器、D 触发器等，触发器是构成时序电路的基本部件，也是计算机中最常用的记忆逻辑电路。

基本 R-S 触发器(Filp Flop)的组成：它可由 2 个与非门(或者 2 个或非门)交叉连接而成，如图 2.28 所示，可存放 1 位二进制信息。输出端 Q 的值与存放的二进制信息对应，$\overline{Q}$ 为反相输出端，输入 $\overline{S}$(Set)为置位端，另一输入 $\overline{R}$(Reset)为复位端。

由于输入低有效,参照连接图、真值表和波形图很容易分析。

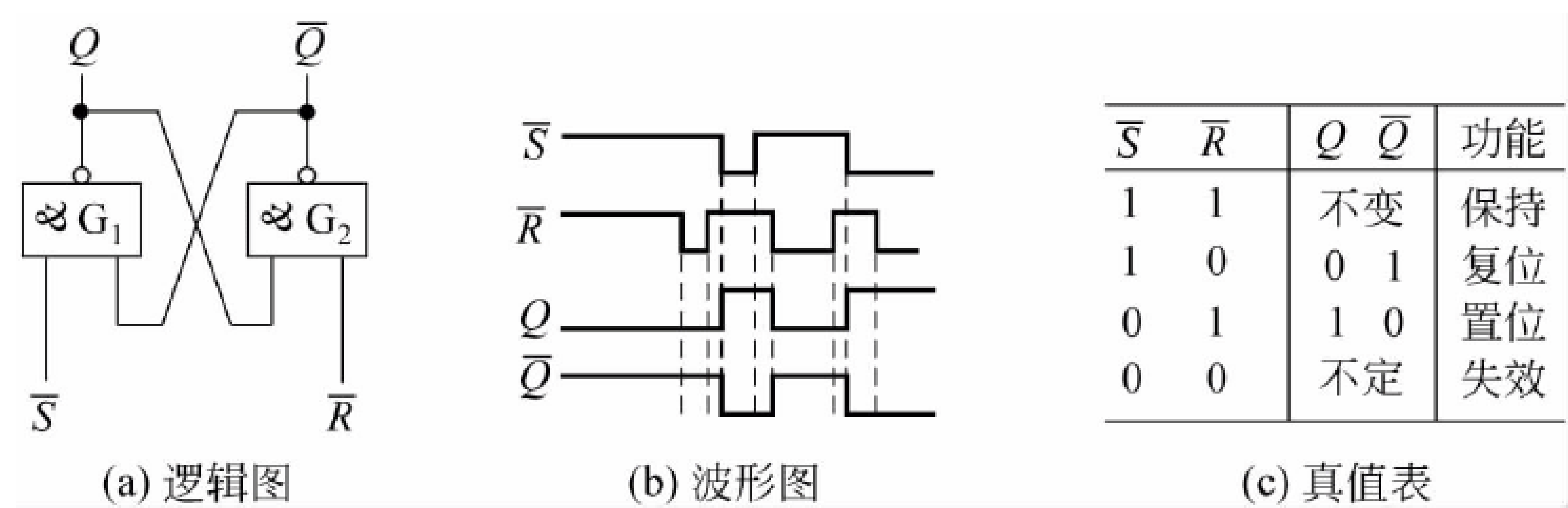


图 2.28 基本 R-S 触发器

可以对 R-S 触发器稍加修改构造出锁存器(Latch)。

在基本 R-S 触发器电路上加 3 个门电路,如图 2.29 所示,使数据接收端 D 仅 1 个,并增加 1 位控制信号 E。E 为低时,触发器状态保持不变,只有 E 为高,输出才能随输入而变。通常称这种触发器为锁存器,在计算机中用于组成暂存器。

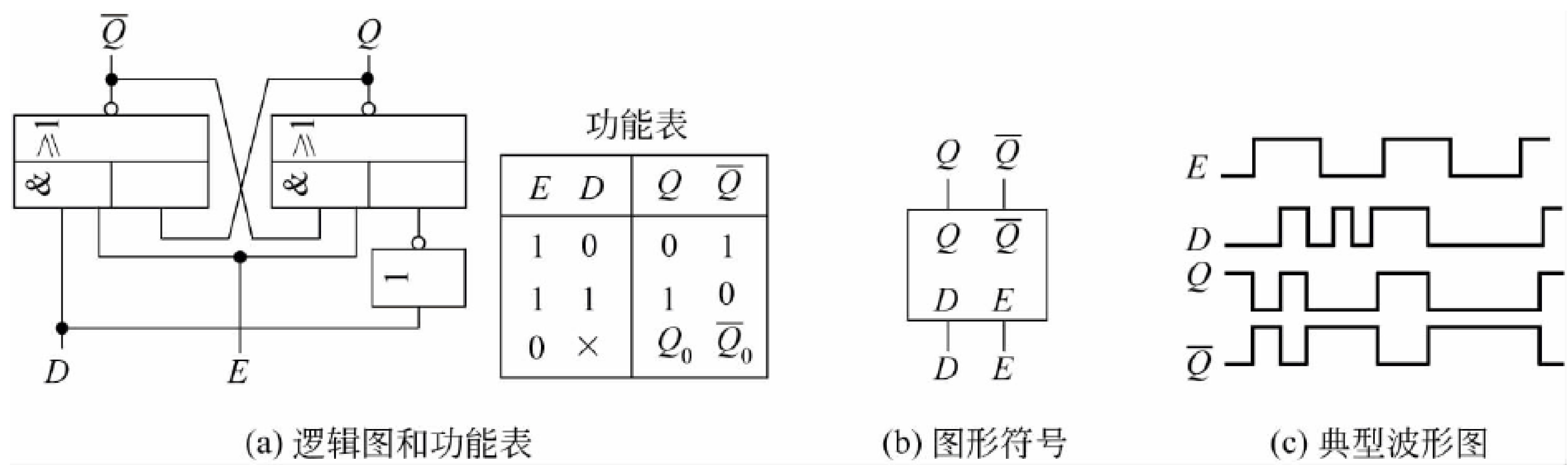


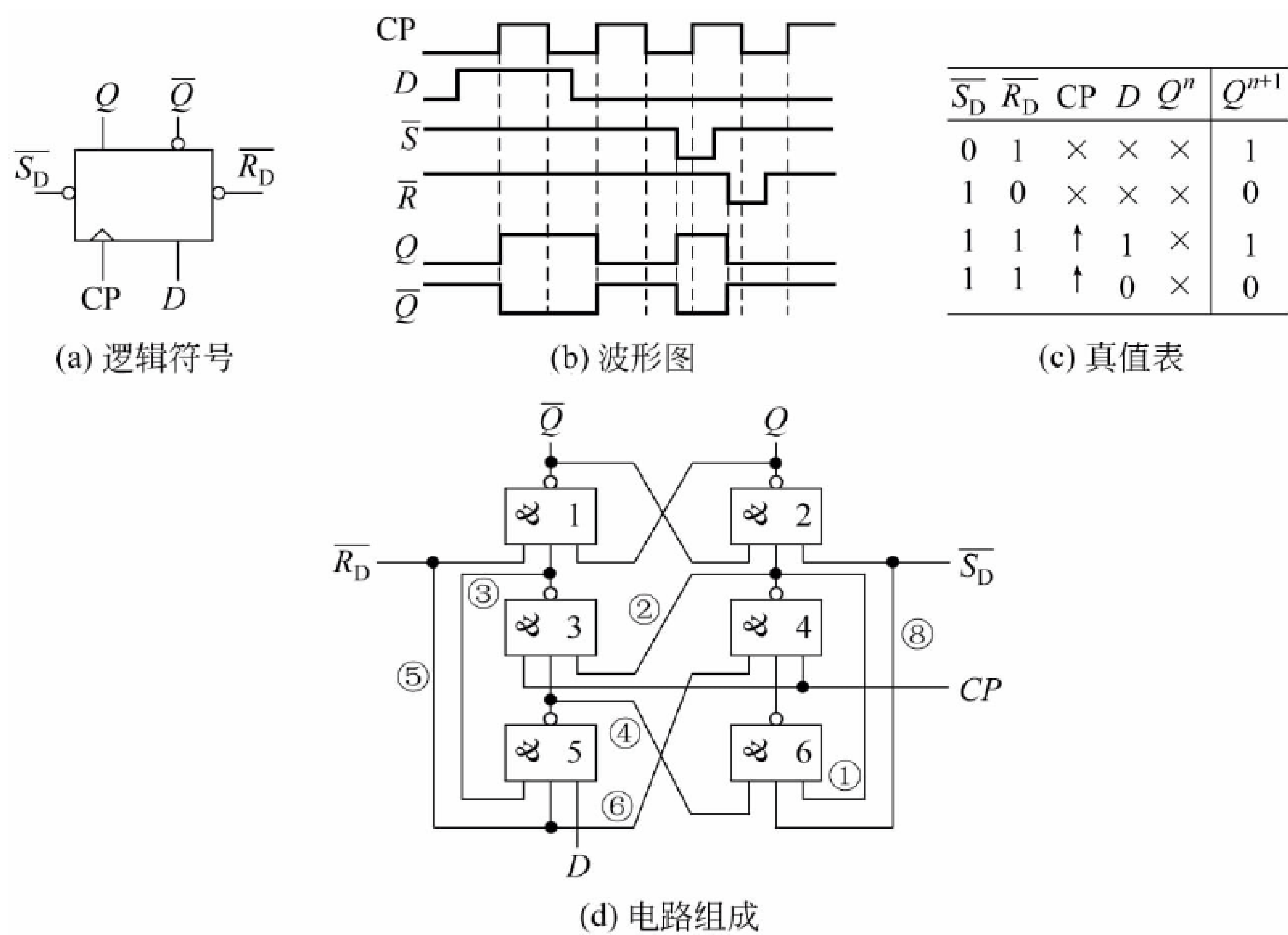
图 2.29 锁存器

2. D 型触发器、寄存器与计数器器件

(1) 基本 R-S 触发器存在不允许 2 个输入端同时为 0、接收数据期间也不允许输入端数据发生变化和没有时钟控制等缺陷。在此基础上再加几个按一定关系连接的与非门就组成比较完善的 D 触发器,如图 2.30 (d) 所示。在 D 型触发器接收输入时自身具有维持阻塞功能,可以保证把自己的输出正常送出并同时接收新的输入数据,这是实现寄存器内容移位和计数功能所需要的。

D 触发器常用于构成计算机中的寄存器和数据缓冲器。它的逻辑符号、激励表和波形图如图 2.30(a)、(b)和(c)所示,其逻辑符号中的 CP 表示时钟脉冲输入信号,D 触发器的状态是在 CP 脉冲的上升沿做相应转换的,即此时输入端 D 的数据传到输出端,Q^n 是它的现态(原来状态),Q^{n+1} 是它的次态(时钟脉冲上升沿作用下将要转换的状态),$\overline{R}_D$ 和 $\overline{S}_D$ 为直接清 0、置 1 端,即不需要时钟脉冲的作用就可以改变触发器的状态,用于对触发器清 0 和预置数据。

D 型触发器可以单独使用,也可以把多个 D 型触发器合成一个部件使用,从而形成接收与发送多位数据的**寄存器**,或带有移位操作功能的**移位寄存器**,或带有计数功能的**进位计数器**等。从实际使用的功能需求考虑,还可以加上对触发器输入和输出控制的功能以及对部件内多位数据共同清"0"控制的功能等。



$\overline{S_D}$	$\overline{R_D}$	CP	D	Q^n	Q^{n+1}
0	1	×	×	×	1
1	0	×	×	×	0
1	1	↑	1	×	1
1	1	↑	0	×	0

图 2.30　D 型触发器

(2) 寄存器(Register)。在教学计算机内部选用了 3 种型号与用途不同的寄存器,它们是 SN74LS377、SN74LS374 和 SN74LS273。

它们的管脚分配基本一致,都有 8 个数据输入和 8 个数据输出;都使用一个时钟输入信号(引脚 11),只是管脚 1 的控制功能有一些差别;内部组成和功能外特性有某些差别。教学计算机系统中,3 种类型的寄存器被使用在不同的场合。SN74LS374 等器件的内部组成和引脚分配如图 2.31 所示。

除此之外,还选用了一片计数器芯片 SN74LS161。这是一个 4 位的二进制同步计数器器件,同步指的是 4 位触发器在同一个时钟脉冲信号作用下同时翻转。

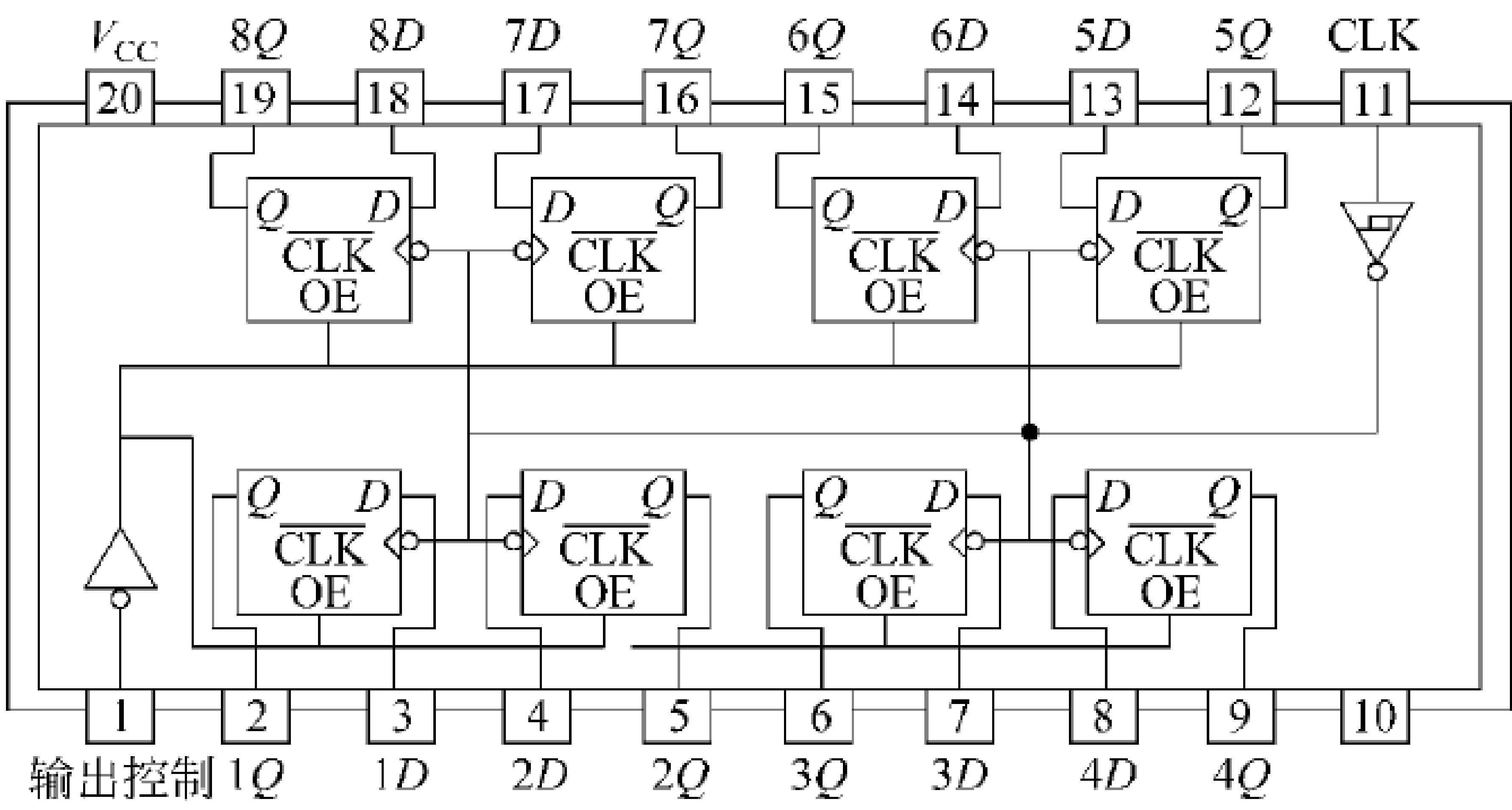
该器件有一个清零输入端 CLR,当它为低电平时(此时两个计数允许控制端都应为低电平状态,置入控制端为高电平状态),时钟脉冲的正跳沿将对 4 个触发器清零。

SN74LS161 有一个预置输入端,当其为低电平时(此时两个计数允许控制端都应为低电平状态,清零控制端为高电平状态),时钟脉冲的正跳沿将把 4 个触发器的输入信号 A、B、C、D 接收到 4 个触发器中,这被称为计数初始状态预置功能。

SN74LS161 还有两个计数允许控制端 P 和 T。当它们为高电平时,且预置输入端、清除输入端也都为高电平,时钟脉冲信号将对 4 个触发器按二进制规则进行计数操作。而当 P 和 T 处于不同状态时,SN74SL161 器件处于禁止运行方式,既不能进行预置操作,也不能进行计数操作。

2.3.3　存储器芯片简介

教学计算机的内存储器由只读存储区和随机读写存储区两部分组成,都选用静态存储器芯片实现。随机读写存储区的容量通常选定为 2K 字,由 2 片随机读写的 2K×8 位的



(a) 内部结构和引脚图

$\overline{G}$	时钟	数据D	Q
H	×	×	Q_0
L	↑	L	L
L	↑	H	H
×	L	×	Q_0

(b) SN74LS377功能表

$\overline{G}$	时钟	数据D	Q
L	↑	H	H
L	↑	L	L
L	L	×	Q_0
H	×	×	Z

(c) SN74LS374功能表

$\overline{G}$	时钟	输入D	Q
L	×	×	L
H	↑	H	H
H	↑	L	L
H	L	×	Q_0

(d) SN74LS273功能表

图 2.31　SN74LS374 等器件的内部组成和引脚

6116 存储器芯片组成。1K＝1024。

只读存储区的容量通常选定为 8K 字,16 位机由 2 片可编程的 8K×8 位的型号为 28C64(或 58C65)的 EEPROM 静态存储器芯片组成。EEPROM 可以用专门的可编程仪器擦除或写入,而教学计算机能够对它直接完成擦除与写入操作。

存储器芯片的内部组成和读写原理将在第 7 章中讲解。图 2.32(c)、(d)给出它们的引脚图,以方便查看。

2.3.4　几个专用功能器件和存储器芯片的引脚图

在教学计算机系统中,在运算器部件中选用了 Am2901 器件,在控制器部件中使选用了 Am2910 器件,在输入/输出接口中选用了串行接口 Intel 8251 和并行接口 Intel 8255 器件。这几个芯片的内部组成和实现的功能都比较复杂,将在用到它们的有关章节中分别讲解。这里给出它们的引脚分配如图 2.32(a)、(b)、(f)、(g)所示。

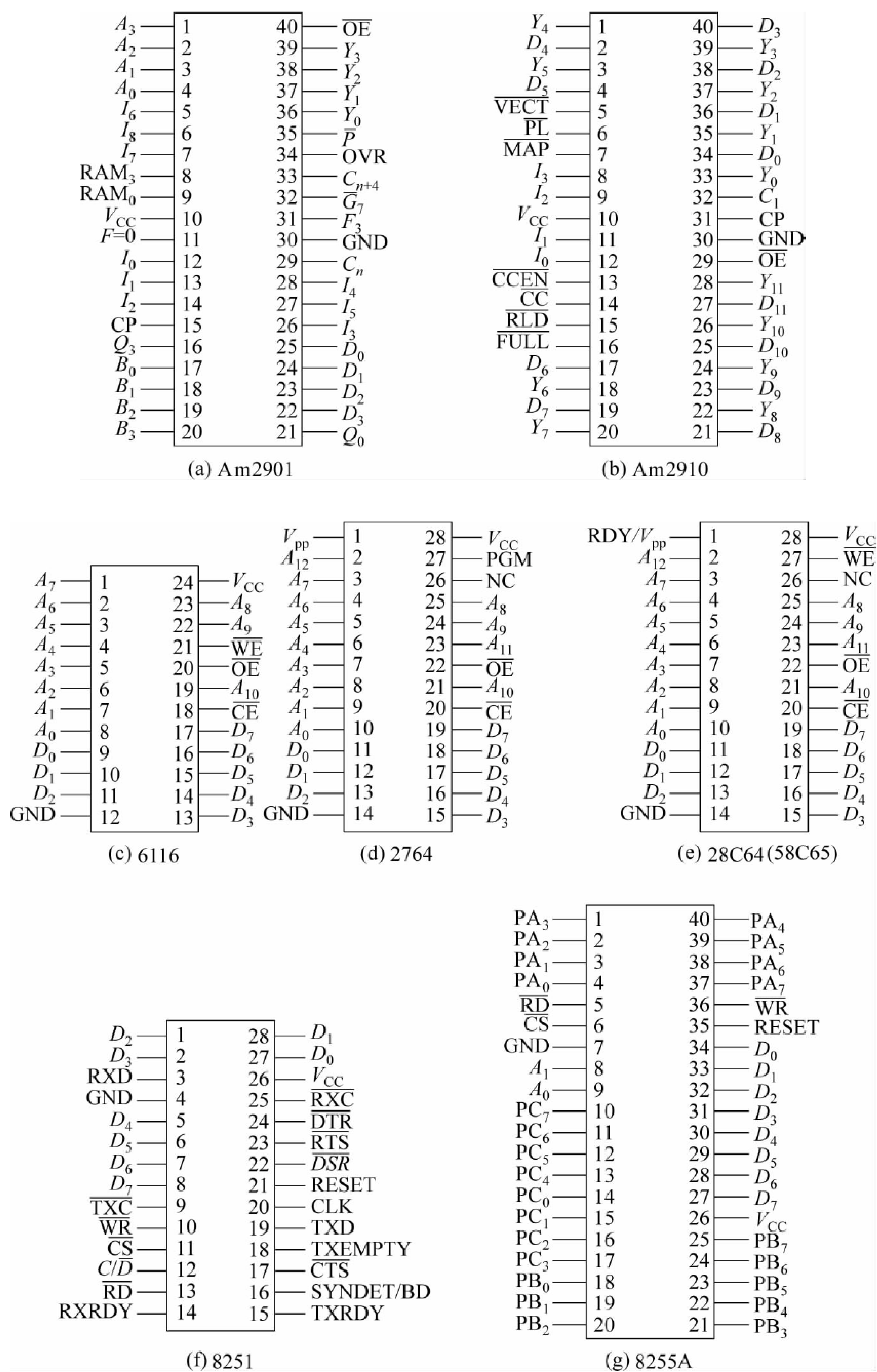


图 2.32　几个专用器件的引脚分配图

2.4 现场可编程逻辑器件及其应用

计算机的发展与微电子技术的发展紧密相关,前述的74系列芯片是标准的器件,用户不能改变其中的逻辑。新型集成电路可编程逻辑器件PLD(Programmable Logic Device),用户可以在此芯片上实现自己的逻辑设计。教学计算机设计中,选用了包括GAL(通用阵列逻辑)、CPLD(Complex PLD)、FPGA(Field Programmable Gate Array)等,用于实现某些组合逻辑电路、时序逻辑电路、控制器中控制信号产生电路CU,甚至完整的CPU系统功能。这些可编程器件对提高教学计算机系统的实验功能发挥着重要的作用。

2.4.1 现场可编程器件概述

PLD芯片中有大量的逻辑门或基本的通用功能模块以及可编程“开关”。通过对这些开关的编程,可以实现逻辑门或功能模块间的不同连接,以实现用户的逻辑电路。

1. GAL的基本原理

以GAL20V8为例介绍,芯片的引脚如图2.33所示,其内部主要由双缓冲、与阵列、输出逻辑宏单元和三态门等组成。

双缓冲是将单一输入转换成原变量和反变量输入的器件。图2.34上方有三个双缓冲。双缓冲在逻辑设计时用到原、反变量时发挥了很重要的作用。GAL20V8芯片最多有20个输入,对应有20个双缓冲。

与阵列也称为乘积项(Product Term),是由与门阵列构成的完成与运算的逻辑电路,如图2.34所示。图2.34中有3个与门,每个与门有6个输入(原、反变量各3个),其中Z_3实现了$Z_3=\overline{A}\cdot\overline{B}\cdot\overline{C}$的逻辑功能。GAL20V8芯片共有8个与门阵列组,每组有8个与门,每个与门有40个输入。

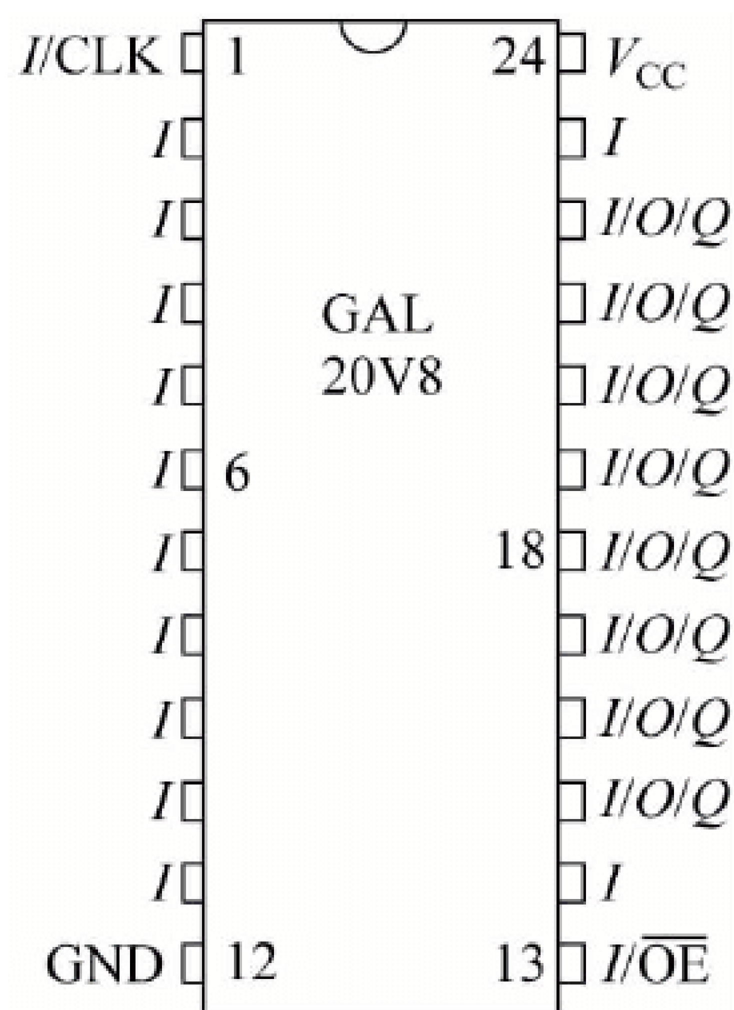


图2.33 GAL20V8引脚图

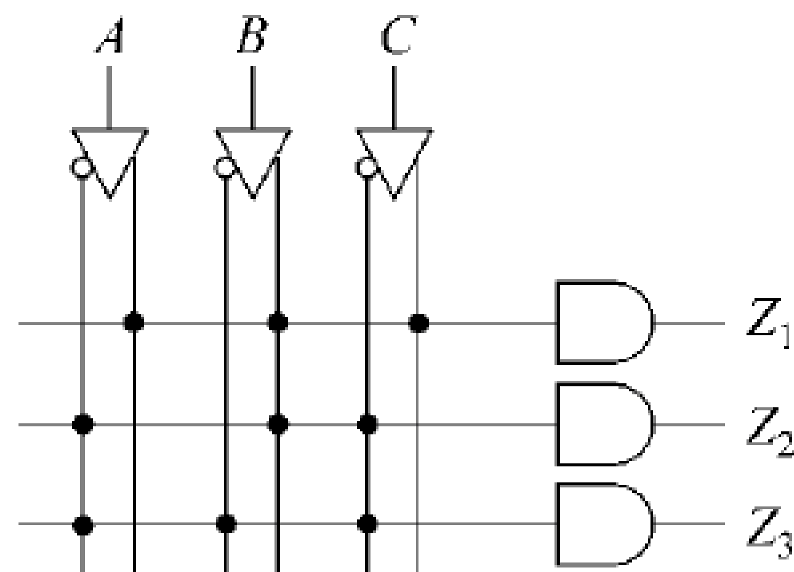


图2.34 与阵列原理图

输出逻辑宏单元(Output Logic Macro Cell)是GAL成为通用器件的关键部件,其内部简化逻辑如图2.35所示,其中的或门用于或运算;异或门用于对或门输出的信号是否反相;D触发器作寄存器;多路控制用于各种灵活的控制;根据实际需要可以将运算后的信息反馈

回输入或将输出引脚作输入用；每路输出都是三态可控的。作时序控制时还可提供时钟信号。

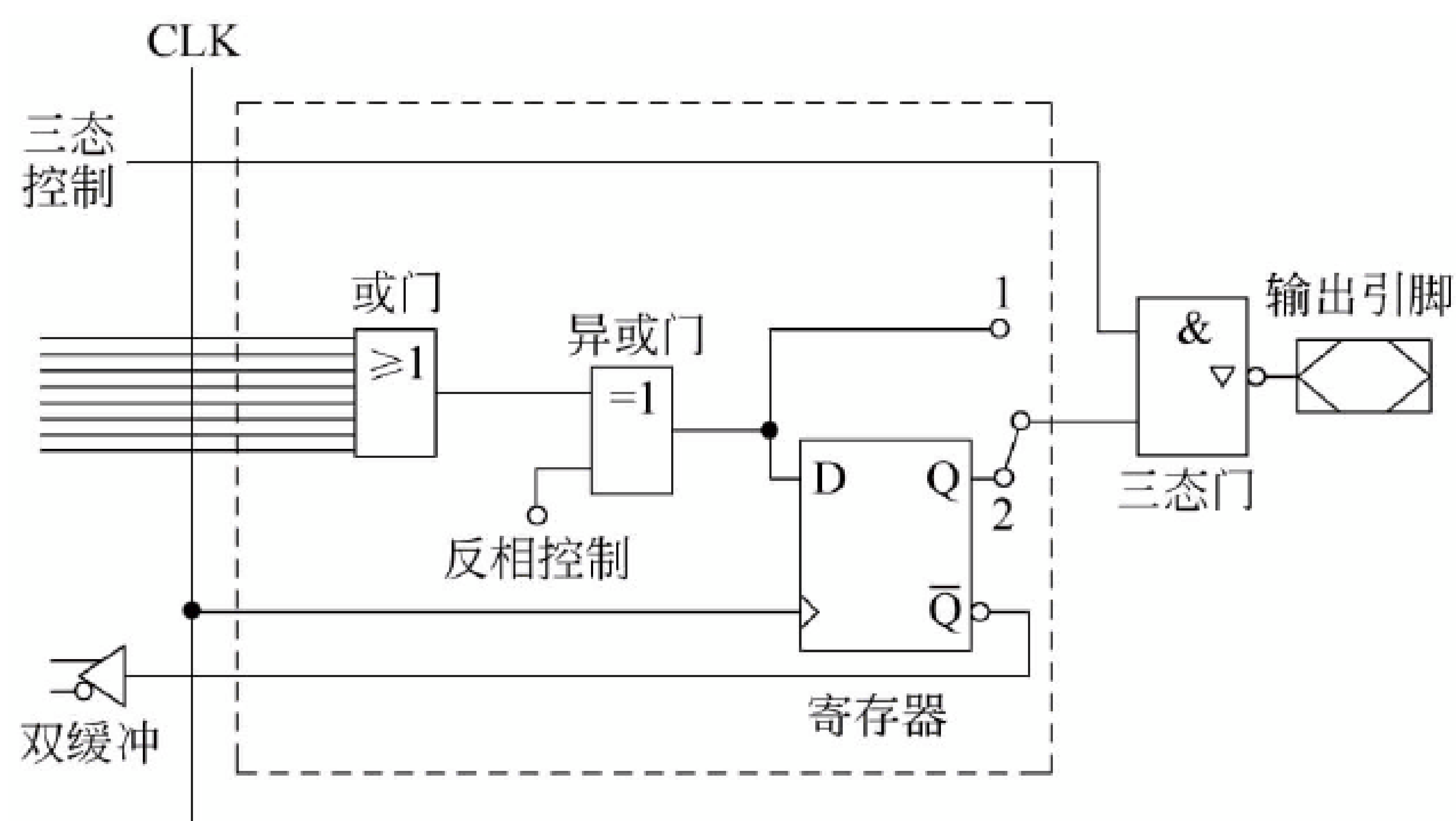


图 2.35　输出逻辑宏单元原理图

2. CPLD 与 FPGA 芯片简介

MACH(Macro Array CMOS High-density)是在 PAL 和 PALCE(相当于 GAL)结构基础上发展起来的复杂的 CPLD 器件，采用 CMOS 电可擦工艺制造，具有连续式的内部连线结构，可以预知内部逻辑的定时关系，容易消除竞争现象。具有通用的设计工具，可选择 ABEL(Advanced Bool Expression Language)或 VHDL(Very High Speed Integrated Circuit Hardware Description Language)语言描述设计对象的组成与功能，并支持在系统编程，关掉电源不会使内部的逻辑信息丢失，便于设计使用。

MACH 系列的高端器件内部由多个优化 PAL 块和一个中央开关矩阵互连而成。其内部的 PAL 块类似于独立的 PAL 器件，它们之间的通信通过中央开关矩阵实现。每个 PAL 块由输入开关矩阵、时钟发生器、乘积项阵列、逻辑分配器、宏单元、输出开关矩阵和 I/O 单元组成。每个 PAL 块内又含有多个宏单元，既有输出宏单元，又有隐埋(Buried)宏单元。中央开关矩阵为 PAL 块的信号输入和块间通信提供通路。

现场可编程门阵列(FPGA)是近些年来出现并开始被广泛应用的大规模集成电路器件。但是其内部结构是采用许多个独立的可编程的逻辑模块(Configurable Logic Block，CLB)、输入输出模块(I/O Block，IOB)和互连资源(Interconnect Resource)3 部分组成。该器件的特点包括以下 5 个方面。

(1) 与 IOB 相连的输入输出引脚数量更多，并可根据需要设置为输入或输出端。

(2) 内部的每一个 CLB 的电路中都包含组合逻辑电路、1～2 个触发器电路和一些数据选择器电路，可根据需要实现组合逻辑的功能，例如 1 个 4 变量的组合逻辑函数，2 个 3 变量的组合逻辑函数，1 个 5 变量的组合逻辑函数。也可以实现时序逻辑的功能，即把触发器编程为边沿触发的 D 触发器，或者电平触发的 D 型锁存器，可以使其运行于同步方式(使用公用的 CLK 时钟信号)或异步方式(使用 1 个特定数据输入端作为时钟)，触发器接收的数据来自组合逻辑部分的输出。

(3) 内部的互连资源由金属线、开关阵列和可编程连接点 3 部分构成，用于实现把数量很大的 CLB 和 IOB 相互连接起来以构成不同的复杂系统。

(4) FPGA 芯片的工作状态(提供的逻辑功能)是由芯片内的编程数据存储器设定，该

存储器中的内容在断电后不被保存,因此必须在每次加电时被重新装入,这是在芯片内的一个时序电路控制下自动完成的。被装入的数据通常要存放在芯片之外的一片 EPROM 器件中。其优点是编程使用更加灵活,缺点是丢掉了数据加密功能。

(5) 可使用更高层次的工具软件,通常可选用 VHDL 语言来描述设计对象的组成与功能(结构与行为)。

例如,Xinlinx 公司的 SPARTAN-Ⅱ系列型号为 XC2S200 的芯片,208 脚的 PQFP 封装形式,有 20 万门容量,芯片内有 2352 个 CLB,可支持在系统编程(In-system Programmable),很适合于用它实现一个简单的 CPU。片内还有 14 个 4Kb 的存储器电路,还可以用其内部的触发器资源构建一个小容量的存储器。

FPGA 结构简化模型由 4 种类型的模块组成:二维逻辑阵列、互连资源、内嵌存储器结构和输入输出等模块,如图 2.36 所示。

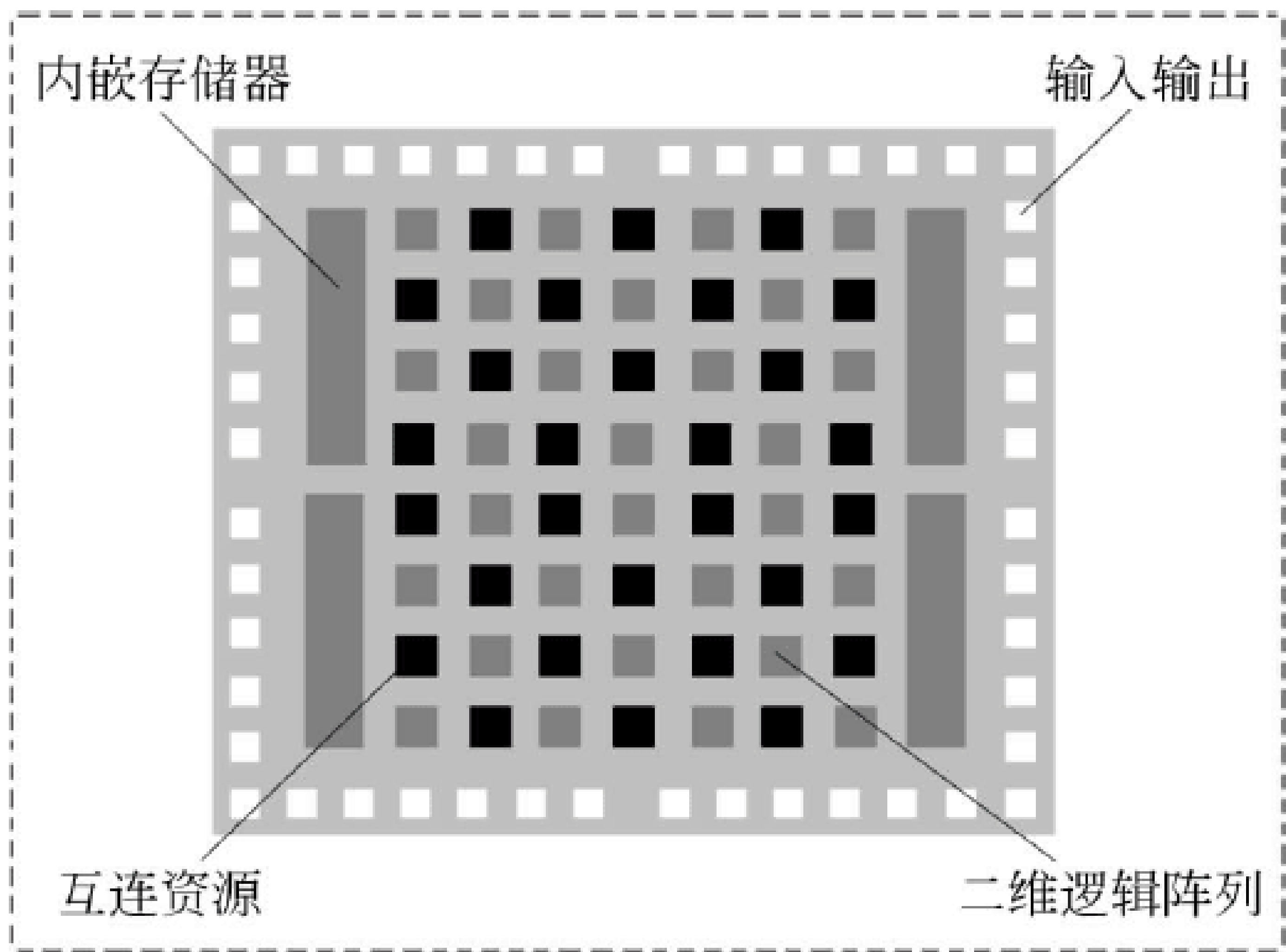
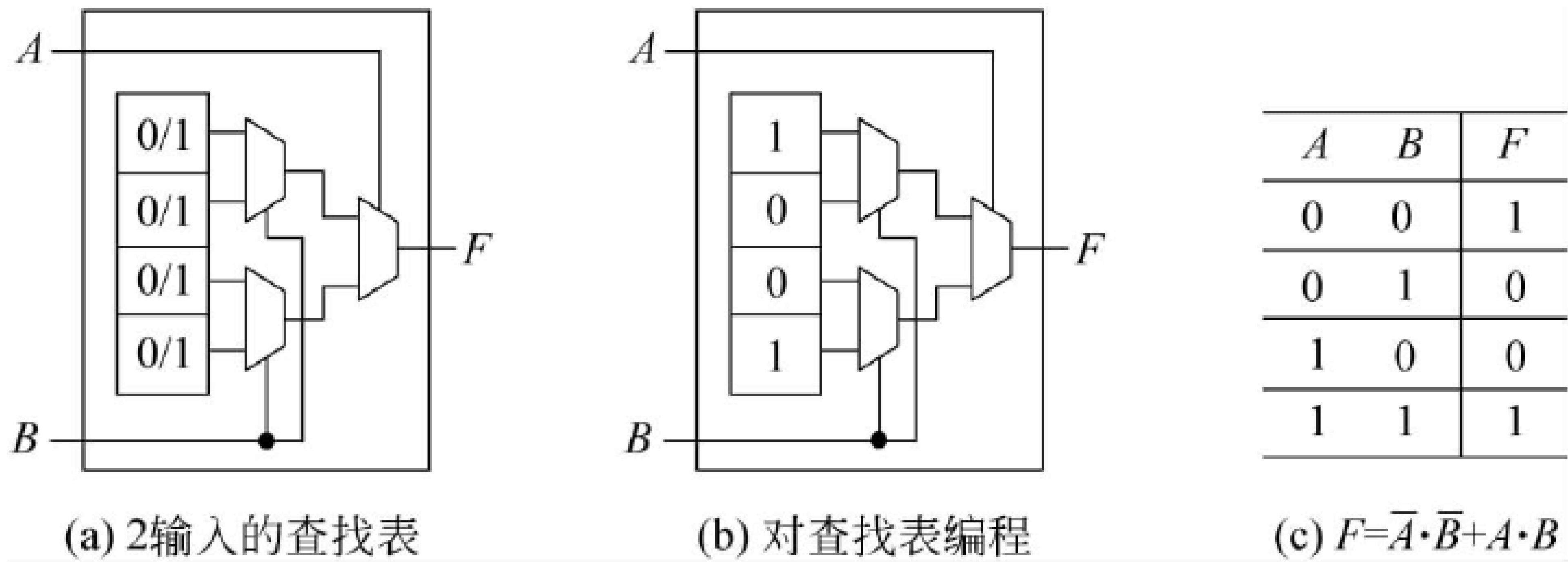


图 2.36　CPLD/FPGA 结构原理图

二维逻辑阵列模块(与、或阵列)是可编程逻辑的主体,用于完成不同的逻辑功能;互连资源模块连接所有的二维逻辑阵列和输入输出模块;内嵌存储器结构可以在芯片内存储数据;输入输出模块是芯片与外界的接口,完成不同要求的输入输出功能。

CPLD 与 FPGA 的内部结构也各不相同,通常 CPLD 基于与、或阵列结构,组合逻辑资源较丰富,虽然也有类似于 OLMC 结构的宏单元,但内部寄存器相对不够丰富,所以较适合设计组合逻辑较多的电路;FPGA 的二维逻辑阵列基于查找表(LookUp Table)结构,如图 2.37 所示,寄存器资源比较丰富,因此也适合设计时序逻辑较多的电路。CPLD 与 FPGA 最大的差别是 CPLD 编程后信息同 ROM 一样,断电后信息仍保留;而 FPGA 编程



A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

图 2.37　FPGA 的查找表结构原理图

后，断电信息即丢失。因此，可以将所需信息（代码或数据）事先存入FPGA芯片外，可选购的EPROM中，在每次开机时自动将所需信息装入FPGA，否则开机时必须通过计算机对FPGA重新下载所需信息。另外，FPGA的优点是编程使用更加灵活，缺点是丢掉了数据加密功能。

2.4.2 CPLD和FPGA的编程与应用

教学计算机中组合逻辑控制器的控制信号产生部件是由MACH（高密度宏阵列CMOS）器件实现的，MACH属于CPLD。

教学计算机中，也用FPGA芯片实现了一个完整的CPU系统。

这两类器件的编程设计大致可分为以下几个步骤。

（1）设计输入。可采用电原理图方式输入，也可用ABEL语言或VHDL等硬件描述语言的文本方式输入。

（2）编译、综合和优化。

（3）功能仿真（或称功能模拟）。通过仿真软件验证设计的正确性。

（4）适配。将设计映射在器件上，进行布局和布线，形成编程用的JEDEC文件。

（5）定时分析和仿真（或称布线后模拟）。利用仿真软件验证器件在要求的频率上能否正常工作。

（6）对器件进行编程。

用于这两类器件的工具软件的使用和设计过程，请参见有关技术资料。

本章内容小结和学习方法建议

本章作为计算机组成原理课的先修知识，每个人应根据自己的实际情况，可以采用不学、适当补修、认真学习等不同的方案处理。主要目标是理解晶体二极管、三极管（双极型）和MOS管的开关特性及其在计算机中的基本应用；理解基本逻辑门及其基本逻辑电路的基本描述或表示方法，并在常用公式和基本规则的基础上，进行简单逻辑设计、逻辑化简等；掌握组合逻辑电路和时序逻辑电路的特点与区别；了解常用组合逻辑电路与常用时序电路，以及最常用的中小规模逻辑电路在计算机组成，特别是其在教学计算机中的应用；了解现场可编程逻辑器件的内部组成和使用特性，了解其简单编程和设计过程。

本章的内容不属于计算机组成原理课程的教学范围，但缺少这部分知识要学懂计算机组成原理课程是非常困难的，特别是涉及计算机各个部件的具体组成和运行原理的实际例子，完成各项硬件实验是一定要对基本元器件有基本的认识和理解，并且会使用它们构成不同规模的功能部件，才可以学习到有实用价值的知识和开展硬件实际工作的能力。

习题与思考题

1. 说明数字逻辑电路与半导体材料是什么关系。
2. 画出二极管、双极型三极管和MOS型三极管理想状态时的等效开关和相应电路。
3. 图2.38为4×4的MOS只读存储器，X为字线（输入），O为位线（输出），试分析当

X_3 字线为 1 时输出 O 对应的十进制数据。

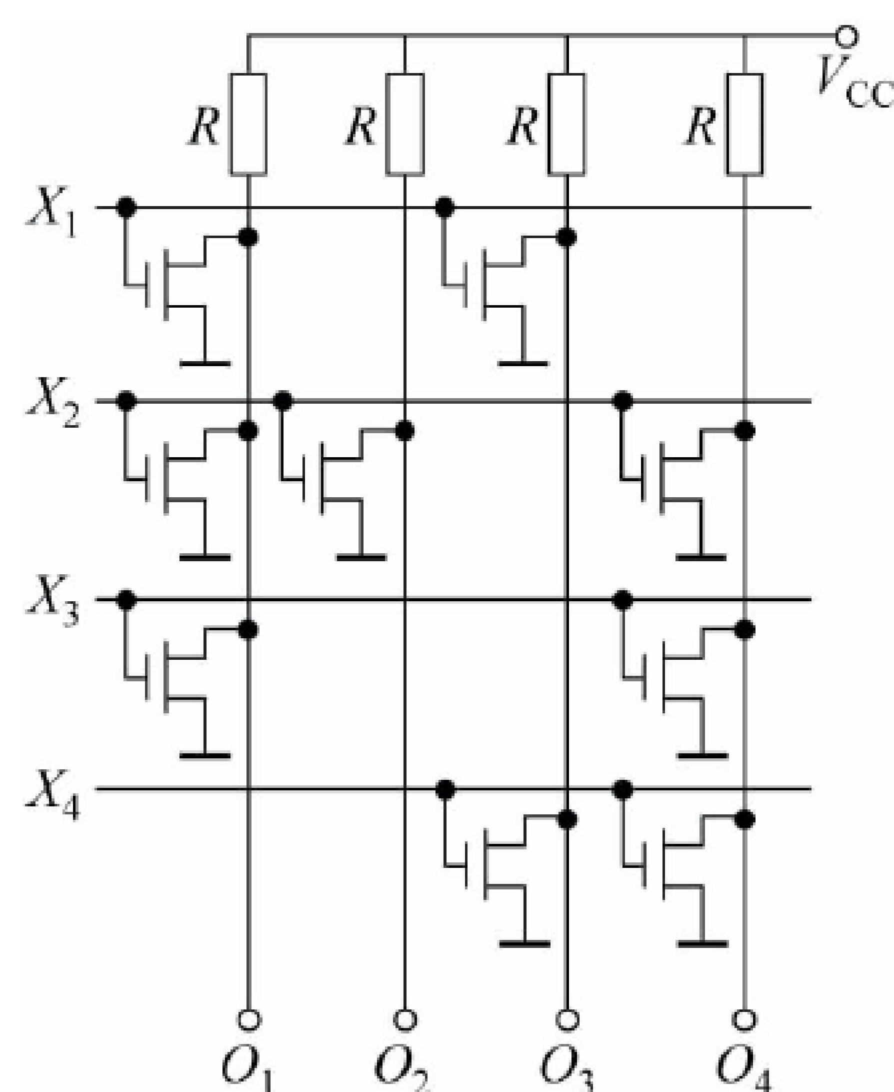


图 2.38 MOS 只读存储器

4. 指出下列 F_1、F_2、F_3 和 F_4 各逻辑表达式中与项的个数和变量的个数。

$F_1=AB+AC+BC$

$F_2=AB+\overline{ABC}$

$F_3=AB+\overline{AB}+\bar{C}$

$F_4=1$

5. 证明 $AB+\bar{A}C+BC=AB+\bar{A}C$。

6. 运用 $AB+\bar{A}C=AB+\bar{A}C+BC$,进行配项,证明

$$AD+B\bar{D}+AB+BD+\bar{A}C+ACEF+\bar{B}EF+DEFG=A+C+BD+\bar{B}EF$$

7. 只使用与、或、非门电路就可以设计出任何复杂功能的组合逻辑、时序逻辑电路的说法正确吗?为什么?

8. 时序逻辑电路和组合逻辑电路的差异表现在哪些方面?造成这种差异的原因是什么?

9. CPLD 器件和 FPGA 器件在内部组成和使用方法等方面有什么区别?

10. 出于什么目的要在本教材中加入本章的内容?

第3章 数据表示、运算算法和线路实现

运算器最重要的功能是加工数据，为此必须很好地掌握各种基本类型的数据及其在计算机内的表示和存储方式，以及完成运算所用的算法和实现这些算法所用的原理性电路。这些内容又以数字化信息编码、布尔代数、数字电路与逻辑设计为基础。布尔代数、数字电路与数字逻辑设计在本教材第2章进行了简要介绍。因此，本章将从数字化信息编码讲起，引出二进制编码，数制转换，插入部分检错纠错码知识；接下来介绍各种基本类型数据的表示；数值数据算术运算的有关方法。

二进制编码，数制转换，定点小数和整数的原码、反码、补码表示是本章的核心重点内容。

检错纠错码概念，讲到的两种常用的检错、纠错码的实现原理应较好理解，而对它们所用的具体线路简单了解即可。

应该掌握定点小数、整数、浮点数在计算机内的表示，补码加减法的运算规则，原码一位乘除法的原理性算法和完成算术运算用到的原理性逻辑线路。补码乘除法、加速乘除运算的可行方案简单了解即可。

本章作业较多，对深入理解所学知识和比较熟练地完成必要的计算很有必要，也是更好地学习运算器知识的基础，应认真完成。

3.1 数字化信息编码的概念和二进制编码知识

3.1.1 数字化信息编码的概念

计算机最重要的功能是处理信息，如数值、文字、符号、语音、图形和图像等。在计算机内部，各种信息都必须采用数字化的形式被保存、加工与传送。掌握信息编码的概念和技术是至关重要的。

所谓编码，就是用**少量、简单的基本符号**，选用**一定的组合规则**，以表示大量复杂多样的信息。基本符号的种类和这些符号的组合规则构成编码的两大要素。例如，用10个阿拉伯数字表示数值，用26个英文字母构成英文词汇，就是现实生活中编码的典型例子。

当要使用的基本符号数量较多时，往往还要采取措施，以便首先使用更少量的简单符号来编码以表示那些量大而复杂的基本符号，再用这些基本符号来表示信息，这就构成了多重编码。多重编码的典型例子是汉字编码，若把上万个汉字都作为基本符号就太多了，可以首

先用笔形字画、偏旁部首、拼音或其他方式对其进行编码,以解决汉字输入问题。

在计算机中,广泛采用的是仅用0和1两个基本符号组成的基二码,亦称为二进制码。主要有以下3个方面的原因。

(1) 基二码在物理上最容易实现,即容易找到具有两个稳定状态且能方便地控制状态转换的物理器件;可以用两个状态分别表示基本符号0和1。

(2) 用基二码表示的二进制数,其编码、计数和算术运算规则简单,容易用数字电路实现,为提高计算机的运算速度和降低实现成本奠定了基础。

(3) 基二码的两个基本符号0和1能方便地与逻辑命题的“否”和“是”,或称“假”和“真”相对应,为计算机中的逻辑运算和程序中的逻辑判断提供了便利条件。

计算机中的各种类型的数据,通常都是用二进制编码形式来表示、存储、处理和传送的。

3.1.2 二进制编码和码制转换

1. 数制与进位计数法

在采用进位计数的数字系统中,如果只用 r 个基本符号(例如 $0,1,2,\cdots,r-1$),通过排列起来的符号串表示数值,则称其为基 r 数制(Radix-r Number System),r 被称为该数制的基(Radix)。假定用 $m+k$ 个自左向右排列的符号 $D_i(-k\leqslant i\leqslant m-1)$ 表示数值 N,即

$$N = D_{m-1}D_{m-2}\cdots D_1D_0D_{-1}D_{-2}\cdots D_{-k} \tag{3.1}$$

式中的 $D_i(-k\leqslant i\leqslant m-1)$ 为该数制采用的基本符号,可取值 $0,1,2,\cdots,r-1$,小数点位置隐含在 D_0 与 D_{-1} 位之间,则 $D_{m-1}\cdots D_0$ 为 N 的整数部分,$D_{-1}\cdots D_{-k}$ 为 N 的小数部分。

如果每个 D_i 的单位值都赋以固定的值 W_i,则称 W_i 为该位的权(Weight),此时的数制称为有权的基 r 数制(Weighted Radix-r Number System)。此时 N 代表的实际值可表示为

$$N = \sum_{i=-k}^{m-1} D_i \times W_i \tag{3.2}$$

如果该数制编码还符合“逢 r 进位”的规则,则每位的权(简称位权)可表示为

$$W_i = r^i$$

式中的 r 是数制的基,i 是位序号。式3.2又可以写为

$$N = \sum_{i=-k}^{m-1} D_i \times r^i \tag{3.3}$$

式中的符号:

r——是这个数制的基;

i——表示这些符号的排列次序,即位序号;

D_i——是位序号为 i 的一位上的符号;

r^i——是第 i 位上的一个1所代表的值(位权);

$D_i\times r^i$——是第 i 位上的符号所代表的实际值;

$\sum$—— 表示对 $m+k$ 位的各位的实际值执行累加求和;

N——代表一个数值。

此时该数制被称为 r 进位数制(Positional Radix-r Number System),简称 r 进制。下面是计算机中常用的几种进位数制:

二进制 $r=2$,基本符号 0,1

八进制 $r=8$，基本符号　0，1，2，3，4，5，6，7

十六进制 $r=16$，基本符号　0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F

其中 A～F 分别表示十进制数 10，11，12，13，14，15

十进制 $r=10$，基本符号　0，1，2，3，4，5，6，7，8，9

如果每一数位都具有相同的基，即采用同样的基本符号集来表示，则称该数制为固定基数制（Fixed Radix Number System），这是计算机内普遍采用的方案。在个别应用中，也允许对不同的数位或位段选用不同的基，即混合采用不同的基本符号集来表示，则该数制被称为混合基数制（Mixed Radix Number System）。

2. 二进制编码和二进制数据

二进制编码是计算机内使用最多的码制。它只使用两个基本符号 0 和 1，并且通过由这两个符号组成的符号串来表示各种信息（各种类型的数据）。二进制的数值类型的数据亦是如此，计算其所代表的数值的运算规则是

$$N=\sum_{i=-k}^{m-1} D_i \times 2^i \qquad D_i \text{ 的取值为 0 或 1} \tag{3.4}$$

例如　$(1101.0101)_2=(13.3125)_{10}$。

等号左右两边括号内的数字为两个不同进制的数字，括号右下角的 2 和 10 分别指明左右两边的数字为二进制和十进制的数。按式(3.4)，计算二进制数 1101.0101 的实际值为

$$1\times 2^3+1\times 2^2+0\times 2^1+1\times 2^0+0\times 2^{-1}+1\times 2^{-2}+0\times 2^{-3}+1\times 2^{-4}$$
$$=8+4+1+0.25+0.0625=13.3125$$

从式中可以进一步看到，由于二进制只用 0 和 1 两个符号，在计算二进制位串所代表的实际值时，只需把符号为 1 的那些位的位权相加即可，则上式变为：

$$2^3+2^2+2^0+2^{-2}+2^{-4}=13.3125$$

熟练地记清二进制数每位上的位权是有益的。当位序号为 0～12 时，其各位上的位权分别为 1、2、4、8、16、32、64、128、256、512、1024、2048 和 4096。

3. 数制转换

计算机中常用几种不同的进位数制，包括二进制、八进制、十进制和十六进制。二进制数据更容易用逻辑线路处理，更接近计算机硬件能直接识别和处理的数字信息的使用要求，而使用计算机的人更容易接受十进制的数据类型。二者之间的进制转换是经常遇到的问题。

1）十进制数据与其他进制数据的转换

式(3.3)确定的运算规则，是不同进位计数制数据之间完成进位制转换的依据。

十进制到二进制的转换，通常要区分数的整数部分和小数部分，并分别按除 2 取余数部分和乘 2 取整数部分两种不同的方法来完成。

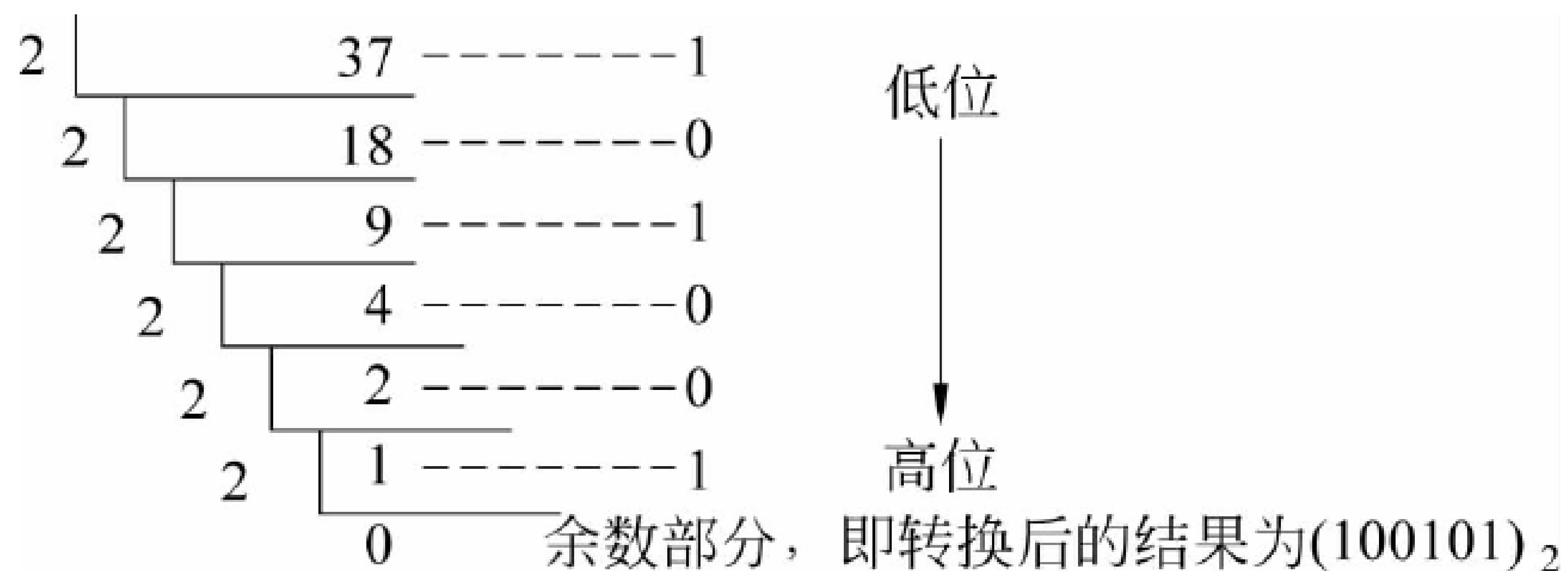
（1）**对整数部分**，要用除 2 取余数办法完成十进制到二进制的转换，其规则是：

① 用 2 除十进制数的整数部分，取其余数为转换后的二进制数整数部分的低位数字；

② 再用 2 去除所得的商，取其余数为转换后的二进制数高一位的数字；

③ 重复执行第②步的操作，直到商为 0，结束转换过程。

例如，将十进制的 37 转换成二进制整数的过程如下：



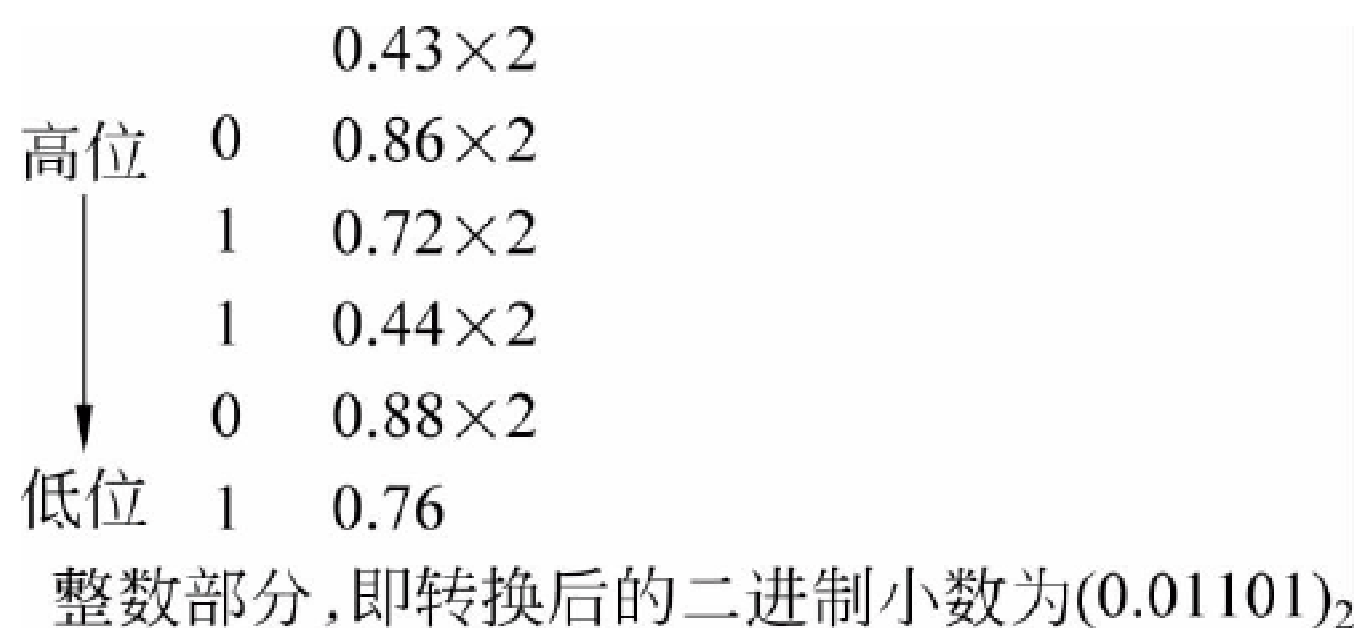
(2) **对小数部分**,要用乘2取整数办法完成十进制到二进制的转换,其规则是:

① 用2乘十进制数的小数部分,取乘积的整数为转换后的二进制数的最高位数字;

② 再用2乘上一步乘积的小数部分,取新乘积的整数为转换后二进制小数的低一位数字;

③ 重复第②步操作,直至乘积部分为0,或已得到的小数位数满足要求,结束转换过程。

例如,将十进制的0.43,转换成二进制小数的过程如下(假设要求小数点后取5位):



对小数进行转换的过程中,转换后的二进制已达到要求位数,而最后一次的乘积的小数部分不为0,会使转换结果存在误差,其误差值小于求得的最低一位的位权。

对既有整数部分又有小数部分的十进制数,可以先转换其整数部分为二进制数的整数部分,再转换其小数部分为二进制的小数部分,通过把得到的两部分结果合并起来得到转换后的最终结果。例如,$(37.43)_{10}=(100101.01101)_2$。

在实现手工转换时,如果对二进制数已经比较熟悉,基本上记住了以2为底的指数值,即二进制数每一位上的权,对十进制数进行转换时,也可以不采用上述规则,基本上可以直接写出来。例如:

$(45.625)_{10}=32+8+4+1+0.5+0.125$

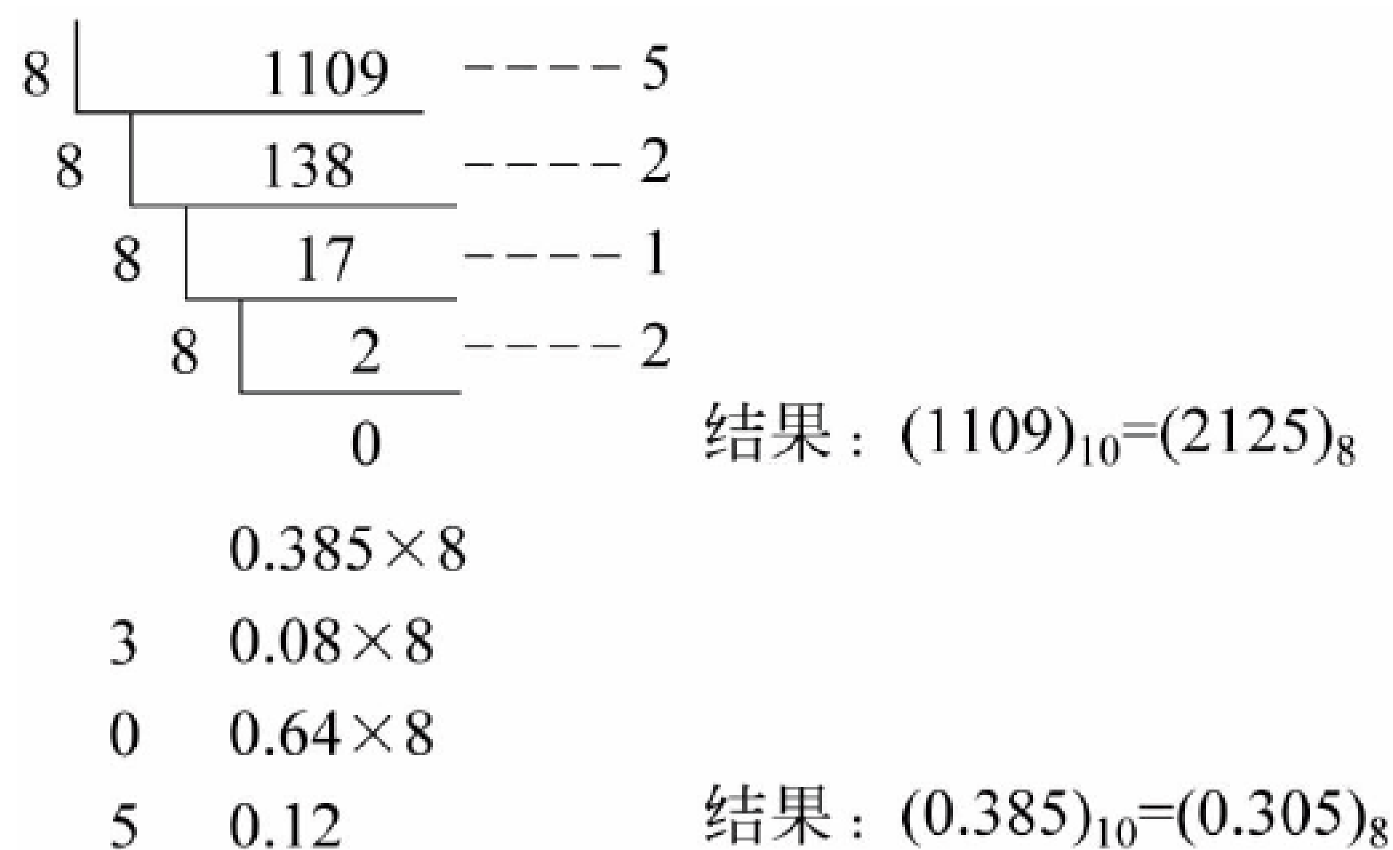
$=(10\ 1\quad 1\ 01.\quad 10\qquad 1)_2$,即$(101101.101)_2$。

$(1105)_{10}=1024+81=1024+64+16+1$

$=(10001010001)_2$,即$(10001010001)_2$。

参照上述方法,也可以实现十进制到八进制、十进制到十六进制的转换过程。例如:

(1) 完成八进制和十进制数据之间的转换,具体如下。



(2) 完成十进制到十六进制数的转换方法与前述方法类似,只是乘除16时,手工运算不大方便。

2) 二进制与八进制、十六进制间的转换

用二进制表示一个数值 N,所用的位数 K 为 $\log_2 N$,如表示4096,K 为13,写起来位串较长。为此,计算机中也常常采用八进制和十六进制来表示数值数据,为表示数值 N,分别有如下对应关系:

$$N = \sum_{i=-k}^{m-1} D_i \times 8^i \qquad D_i \text{ 的取值为 0 到 7} \tag{3.5}$$

例如 $(7.44)_8 = 7\times8^0 + 4\times8^{-1} + 4\times8^{-2} = (7.5625)_{10}$。

$$N = \sum_{i=-k}^{m-1} D_i \times 16^i \qquad D_i \text{ 的取值为 0 到 9 和 } A \text{ 到 } F \tag{3.6}$$

例如 $(1A.08)_{16} = 1\times16^1 + 10\times16^0 + 8\times16^{-2} = (26.03125)_{10}$。

上述二式中所用符号的意义与式(3.3)中所用符号的意义类同,但此处 D_i 包含的基本符号分别限于0~7和0~9、A~F,各位的码权分别为 8^i 和 16^i。

把用二进制、八进制、十六进制表示的数转换成十进制数的值,使人能更容易地衡量这个数值的大小。

由于 $\log_2 8=3$,$\log_2 16=4$,即1位八进制的数可以用3位二进制的数重编码来得到,1位十六进制的数可以用4位二进制的数重编码得到,故人们通常认为,在计算机这个领域内,八进制和十六进制数只是二进制数的一种特定的表示形式。表3.1给出了少量二进制、八进制、十六进制和十进制数的对应关系。

表3.1　二进制、八进制、十六进制和十进制的对应关系

二进制数	八进制数	十六进制数	十进制数的值
0000	00	0	0
0001	01	1	1
0010	02	2	2
0011	03	3	3
0100	04	4	4
0101	05	5	5
0110	06	6	6
0111	07	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12

续表

二进制数	八进制数	十六进制数	十进制数的值
1101	15	D	13
1110	16	E	14
1111	17	F	15

在把二进制数转换成八进制或十六进制表示时，应从小数点所在位置分别向左、向右对每3位或每4位二进制位进行分组，写出每一组所对应的1位八进制数或十六进制数。若小数点左侧(即整数部分)的位数不是3或4的整数倍，可以按在数的最左侧补零的方法理解；对小数点右侧(即小数部分)，应按在数的最右侧补零的方法处理，否则容易转换错。对不存在小数部分的二进制数(整数)，应从最低位开始向左把每3位划分成一组，使其对应一个八进制位，或把每4位划分成一组，使其对应一个十六进制位。例如，$(10.101)_2$变成八进制时，应把它理解为$(010.101)_2$，是$(2.5)_8$，即八进制的2.5。当把它转换为十六进制时，应首先变为$(0010.1010)_2$，是$(2.A)_{16}$，即十六进制的2.A，而不是$(2.5)_{16}$。又如：

$$(1100111.10101101)_2=(147.532)_8$$

$$(1100111.10101101)_2=(67.AD)_{16}$$

八进制和十六进制之间的转换不是很常用，二者经过二进制的中间结果进行转换是方便的。

4. 二进制数的运算规则

二进制数之间可以执行算术运算和逻辑运算，其规则简单，容易实现。

1）加法运算规则

```
0+0=0              例如：  1101
0+1=1                   +) 1001
1+0=1                   -------
1+1=0    (产生进位)        10110
```

2）减法运算规则

```
0-0=0              例如：  1101
0-1=1   (产生借位)       -) 0111
1-0=1                   -------
1-1=0                      0110
```

3）乘法运算规则

二进制数乘法的计算方法，与十进制数乘法的计算方法类似，按乘数每一位的值计算出部分积，对全部部分积求累加和则得到最终乘积。

```
             例如：  1101
                   ×1001
0×0=0            --------
                     1101
0×1=0               0000
                   0000
1×0=0             1101
                 --------
1×1=1             1110101
```

4）除法运算规则

二进制数除法的计算与十进制数除法类似，也由减法、逐位上商等操作分步完成。例如：

```
           1101
     ___________
1001 /1110101
      1001
     ------
       1011
       1001
      ------
         1001
         1001
        -----
            0
```

逻辑运算是在对应的两个二进制位的逻辑值之间进行的，与相邻的高低位的值均无关，即不存在进位、借位等问题。

5）逻辑或运算规则（运算符为∨）

$0 \vee 0=0$	例如：
$0 \vee 1=1$	1100
$1 \vee 0=1$	∨ 1010
$1 \vee 1=1$	1110

6）逻辑与运算规则（运算符为∧）

$0 \wedge 0=0$	例如：
$0 \wedge 1=0$	1100
$1 \wedge 0=0$	∧ 1010
$1 \wedge 1=1$	1000

7）逻辑非运算规则（运算符为¬）

$$\neg 0=1$$

$$\neg 1=0$$

逻辑非实现对单个逻辑值的处理，而不是对两个逻辑值的运算，逻辑非又被称为逻辑取反操作。对逻辑数 1011 逐位进行取反，其结果为 0100。

8）逻辑异或运算规则（运算符为∀）

0 ∀ 0=0	例如：
0 ∀ 1=1	1100
1 ∀ 0=1	∀ 1010
1 ∀ 1=0	0110

与、或、非操作是三种最基本的逻辑操作，用它们可以组合出任何逻辑运算功能。某些情况下，还要用到逻辑异或操作。逻辑异或实现的是按位加功能，只有参与异或操作的两个逻辑值不同时（一个为 0，另一个为 1），结果才为 1，和或操作结果的差异表现在：或操作中 1 或 1=1，而异或操作则是 1 异或 1=0。

3.1.3 检错纠错码

1. 检错纠错的有关概念和实现思路

数据在计算机系统内加工、存取和传送的过程中可能产生错误。为减少和避免这类错

误,一方面是精心选择各种电路,改进生产工艺与测试手段,尽量提高计算机硬件本身的可靠性;另一方面是在数据编码上找出路,即采用带有某种特征能力的编码方法,通过少量的附加电路,使之能发现某些错误,甚至能准确地确定出错位置,进而提供自动纠正错误的能力。

数据校验码就是一种常用的带有发现某些错误,甚至带有一定自动改错能力的数据编码方法。它的实现原理是在合法的数据编码之间,加进一些不允许出现的(非法的)编码,使合法数据编码出现某些错误时,就成为非法编码。这样,则可以通过检查编码的合法性来达到发现错误的目的。合理地设计编码规则,安排合法或不合法的编码数量,就可以得到发现错误的能力,甚至达到自动改正错误的目的。这里用到一个**码距**(最小码距)的概念。码距是指任意两个合法码之间至少有几个二进制位不相同,仅有一位不同,称其(最小码距)为1,例如用4位二进制表示16种状态,则16种编码都用到了,此时码距为1。就是说,任何一个编码状态的4位码中的一位或几位出错,都会变成另一个合法码,此时无检错能力。若用4个二进制位表示8种合法状态,就可以只用其中的8个编码来表示之,而把另8种编码作为非法编码,此时可能使合法码的码距为2。一般说来,合理地增大编码的码距,就能提高发现错误的能力,但表示一定数量的合法码所使用的二进制位数要变多,增加了电子线路的复杂性和数据存储、数据传送的数量。在确定与使用数据校验码的时候,通常要考虑在不过多增加硬件开销的情况下,尽可能地发现较多的错误,甚至能自动改正某些最常出现的错误。常用的数据校验码是奇偶校验码、汉明校验码、循环冗余校验码等。纠错编码是对检错编码更进一步的发展和应用。

计算机内经常遇到的错误有两大类:随机错误和突发错误。前者指孤立出现的一个错误,后者指连续产生的一批(彼此之间可能有关联)错误。对它们处理的难度和复杂度会有很大不同,在我们的课程中基本不涉及对突发错误的检查与纠正问题,有兴趣者请自行查阅有关资料。纠错编码的分类方案如图3.1所示。

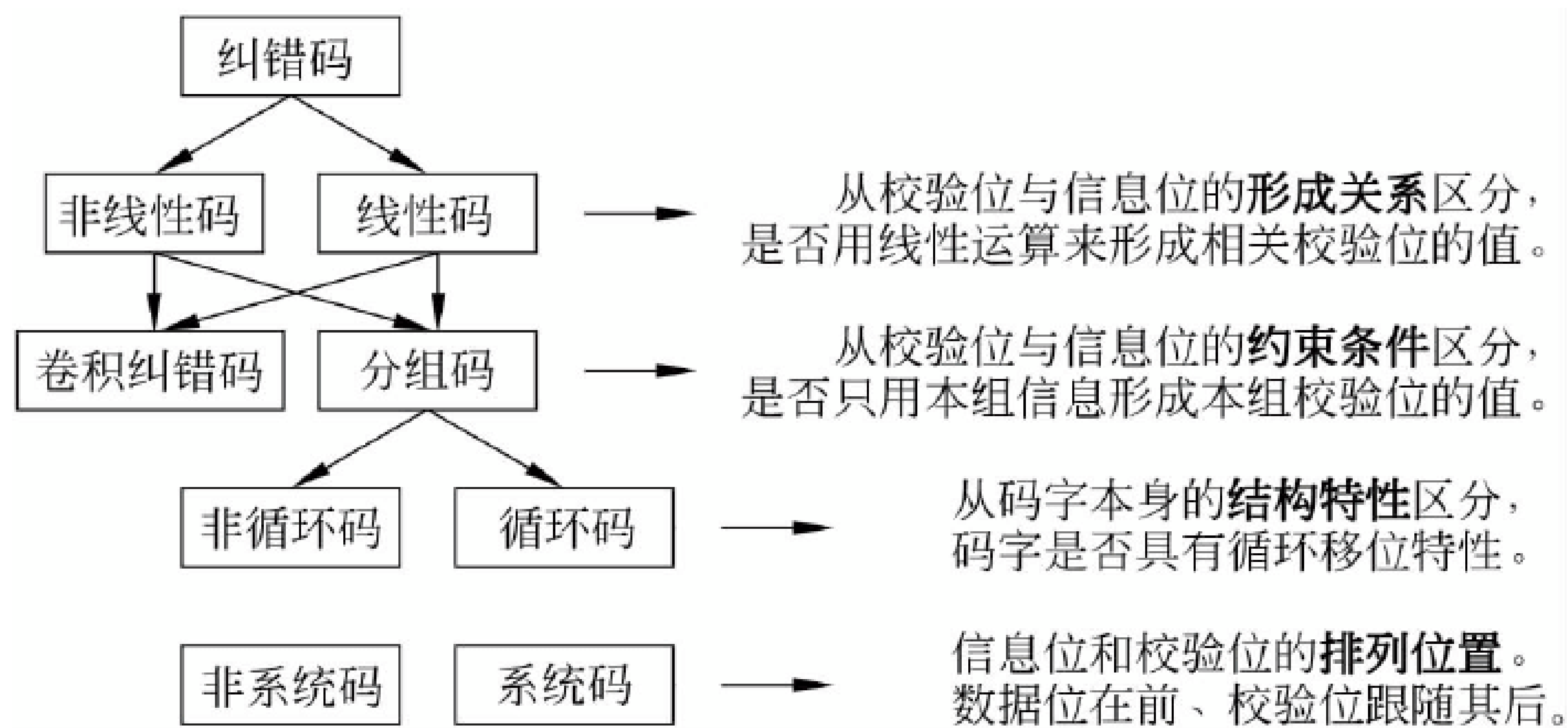


图3.1 纠错码的分类

2. 3种常用的检错纠错码

1) 奇偶校验码

奇偶校验码是一种开销最小,能发现数据代码中一位出错情况的编码,常用于存储器读写检查,或ASCII字符、其他类型信息传送过程中的出错检查。它的实现原理,是使原来合法编码码距由1增加到2。若合法编码中有一个二进制位的值出错了,由1变成

0，或由0变成1，这个码必将成为非法编码。实现的具体方法，通常是为一个字节补充一个二进制位，称为校验位，通过设置校验位的值为0或1的方式，使字节自身的8位和该校验位含有1值的位数一定为奇数或偶数。在使用奇数个1的方案进行校验时，称为奇校验，反之则称为偶校验。依据8位的数据位中为1值的个数确定校验位的值，是由专设的线路实现的，通常使用几级异或门电路产生校验位的值，其输入是数据位的信号或异或运算的中间结果，最后的输出信号是校验位的值。例如，当要把一个字节的值写进主存时，首先用此电路形成校验位的值，然后将这9位的代码作为合法数据编码写进主存。当下一次读出这一代码时，再用相应线路检测读出9位码的合法性。若在主存写进、存储或读出的过程中，某一个二进制位上出现错误，得到的9位码必变成非法编码，从而发现一定是哪一位上出现了错误。这种方案只能发现一位错或奇数个位出错，但不能确定是哪一位错，也不能发现偶数个位出错。考虑到，一位出错的概率比多位同时出错的概率高得多，该方案还是有很好的实用价值。

下面给出对几个字节值的奇偶校验的编码结果：

数据	奇校验的编码	偶校验的编码
00000000	**1**00000000	**0**00000000
01010100	**0**01010100	**1**01010100
01111111	**0**01111111	**1**01111111
11111111	**1**11111111	**0**11111111

该例子中，码字的最高一位为校验位，其余低八位为数据位。从中可以看到，校验位的值取0还是1，是由数据位中1的个数、是奇校验还是偶校验方案共同决定的。

2）汉明校验码

这是由Richard Hamming于1950年提出、目前还被广泛采用的一种很有效的校验方法。其只要增加少数几个校验位，就能检测出两位同时出错、亦能检测出一位出错并能自动恢复该出错位的正确值的有效手段，后者被称为自动纠错。它的实现原理，是在k个数据位之外加上r个校验位，从而形成一个$k+r$位的新的码字，使新得到的码字的码距比较均匀地拉大。若把数据的每一个二进制位合理地分配在几个不同的偶校验位的组合中，使每一个数据位与不同的校验位组合建立准确的对应关系，则当某一位出错后，就会引起相关的几个校验位的值发生变化，这不但可以发现出错，还能指出是哪一位出错，为进一步自动纠错提供了依据。

假设为k个数据位设置r个校验位，则校验位能表示2^r个状态，可用其中的一个状态指出“没有发生错误”，用其余的2^r-1个状态指出有错误发生在某一位，包括k个数据位和r个校验位，因此校验位的位数应满足如下关系：

$$2^r \geqslant k+r+1 \qquad (3.7)$$

如果要求在检出与自动校正一位错的基础上，也能同时发现两位错，此时应该在前一种条件下再增加一位校验位，称为总校验位，专用于区分是一位出错还是双位出错，则此时的校验位的位数r和数据位的位数k应满足下述关系：

$$2^{r-1} \geqslant k+r \qquad (3.8)$$

式(3.8)是通过用$r-1$替代式(3.7)中的r得到的。

按上述不等式，可计算出数据位k与校验位r的对应关系，如表3.2所示。

表3.2 数值位 k 和校验位 r 的对应关系

k 值	最小的 r 值	k 值	最小的 r 值
3～4	4	27～57	7
5～11	5	58～120	8
12～26	6		

设计汉明码编码的关键技术，是合理地把每个数据位分配到 r 个校验组中，以确保能发现码字中任何一位出错；若要实现纠错，还要求能指出是哪一位出错，对出错位求反则得到该位的正确值。例如，当数据位为3位(用 D_3 D_2 D_1 表示)时，检验位应为4位(用 P_4 P_3 P_2 P_1 表示)。可通过表3.3表示的关系，完成把每个数据位划分在形成不同校验位的偶校验值的逻辑表达式中。

表3.3 校验位与数据位的对应关系

	D_3	D_2	D_1	P_4	P_3	P_2	P_1
	1	1	1	**1**	1	1	1
最低3行	1	1	0	0	**1**	0	0
	1	0	1	0	0	**1**	0
	0	1	1	0	0	0	**1**
低3行编码	6	5	3	0	4	2	1

表中 D_3 D_2 D_1 为3位数据位，P_4 P_3 P_2 P_1 为4位校验位，其中低3位中的每一个校验位 P_3 P_2 P_1 的值，都是用3个数据位中不同的几位通过偶校验运算规则计算出来的。其对应关系是：对 P_i(i 的取值为1～3)，总是用处在 P_i取值为1的行中的、用1标记出来的数据位来计算出该 P_i的值。最高一个校验位 P_4，被称为总校验位，它的值是通过对全部3个数据位和其他全部校验位(不含 P_4 本身)执行偶校验计算求得的。

在 P_1、P_2、P_3、P_4 竖列相应行分别填1，在该4列的低3横行其他位置分别填0，在最顶横行的每个尚空位置都分别填1。**若只看低3横行**，右4竖列的3个位的组合值分别为十进制的1、2、4、0，则分配 D_1 D_2 D_3 列的组合值为3 5 6，保证低3横行各竖列的编码值各不相同。

计算各校验位的值的过程叫作编码，按刚说明的规则，4个校验位所用的编码方程为

$$P_4 = D_3 \oplus D_2 \oplus D_1 \oplus P_3 \oplus P_2 \oplus P_1$$
$$P_3 = D_3 \oplus D_2$$
$$P_2 = D_3 \oplus D_1$$
$$P_1 = D_2 \oplus D_1$$

由多个数据位和多个校验位组成的一个码字，将作为一个数据单位处理，例如被写入内存或被传送走。之后，在执行内存读操作或在数据接收端，则可以对得到的码字，通过偶校验来检查其合法性，通常称该操作过程为译码，所用的译码方程为

$$S_4 = P_4 \oplus D_3 \oplus D_2 \oplus D_1 \oplus P_3 \oplus P_2 \oplus P_1$$
$$S_3 = P_3 \oplus D_3 \oplus D_2$$
$$S_2 = P_2 \oplus D_3 \oplus D_1$$
$$S_1 = P_1 \oplus D_2 \oplus D_1$$

译码方程和编码方程的对应关系很简单。译码方程是用一个校验码和形成这个校验码的编码方程执行异或，实际上是又一次执行偶校验运算。通过检查4个S的结果，可以实现检错纠错的目的。实际情况是，当译码求出来的S_4、S_3、S_2、S_1的结果与表3.3中的哪一列的值相同，就说明是哪一位出错；故人们又称表3.3为出错模式表。若出错的是数据位，对其求反则实现纠错；若出错的是校验位则不必理睬。举例如下。

(1) 任何一位(含数据位、校验位)均不错，则4个S都应为0值(思考为什么会如此)。

(2) 任何单独一位数据位出错，4个S中会有3个为1；如D_3错，则S_4 S_3 S_2 S_1为1110。

(3) 若单独一位校验位出错，4个S中会有一个或两个为1；如P_1错，S_4 S_3 S_2 S_1为1001，如P_4错，S_4 S_3 S_2 S_1为1000。

(4) 任何两位(含数据位、校验位)同时出错，S_4一定为0，而另外3个S位一定不全为0，此时只知道是两位同时出错，但不能确定是哪两位出错，故已无法纠错。如D_1和P_2出错，会使S_4 S_3 S_2 S_1为0001。请注意，S_4的作用在于区分出错的位数是奇数还是偶数，S_4为1是奇数个位出错，为0是无错或者偶数个位出错。这不仅为发现两位错所必需，也是为**确保**能发现并改正一位错所必需的。若不设置S_4，某两位出错对几个S的影响与单独另一位出错可能是一样的(不必花费精力推敲)，此时若不加以区分，简单地按一位出错自动完成纠错处理反而会越纠越错。

3) 循环冗余校验码——CRC码

二进制信息位串沿一条信号线逐位地在部件之间或计算机之间传送称为串行传送。CRC(Cyclic Redundancy Check)码可以发现并纠正信息串行读写、存储或传送过程中出现的一位、多位错误。因此，在外存储器设备读写和计算机之间串行通信的场合得到普遍应用。

CRC码一般是指k位信息码之后拼接r位校验码。其特点是一个合法码字实现首尾连接的循环移位，每一步移位操作得到的都是合法码字。应用CRC码的关键是如何从k位信息位简便地得到r位校验位(编码)的值，以及如何判断$k+r$位的码字是否正确。下面仅就CRC码应用中的问题做简单介绍(有关的理论问题请参阅有关书籍)。

(1) 模2运算是指以按位模2相加为基础的四则运算，运算时不考虑位间进位和借位。

(2) **CRC码编码**是经过模2除运算来完成的。其得到商的规则是：当被除数或者部分余数的最高位为1时，商1；为0时，商0。当余数的位数小于除数的位数时结束运算，该余数即为校验位的值。实现这种运算用到的线路很简单，实现左移的移位寄存器(实现串行输出)加上与除数有关的异或控制门即可完成，关键是如何得到模2除运算的除数(称为生成多项式)，通常可去查数学表。

(3) **CRC的译码与纠错**，在数据接收端，用约定的生成多项式去除收到的循环码，如果码字无误则余数应为0，如有某一位出错，则余数不为0，且不同数位出错余数会不同，余数

与出错位的对应关系是不变的,只与码制和生成多项式有关。还用这个电路,再经过几步循环移位和相关操作即可实现纠错。

3.2 数据表示

在这一节,重点讲解数据表示,即计算机内最常用的信息编码。其包括逻辑型数据表示、中西文字符编码表示、数值型数据的编码表示。这些内容是数字计算机设计与应用的基本知识之一,要求能熟练掌握并运用自如。

3.2.1 逻辑类型数据的表示

逻辑数据是用来表示二值逻辑中的“是”与“否”或称“真”与“假”两个状态的数据。很容易想到,用计算机中的基2码的两个状态1和0恰好能表示逻辑数据的两个状态。例如用1表示真,则0表示假。请注意,这里的1和0没有了数值有无或大小的概念,只有逻辑上的意义。在计算机内,可以用一位基2码表示逻辑数据,就是说,8个逻辑数据可以存放在1个字节中,可用其中的每一位表示一个逻辑数据。

正确地建立逻辑数据的概念是十分重要的。因为计算机内部的所有数字化信息,不论它们是数据、指令或控制信号,都是用基2码表示的,其存储与处理都是用逻辑线路实现的。可以说,逻辑线路是计算机硬件组成的基石,而逻辑线路处理的对象就是逻辑型数据,通称逻辑变量,即实现输出逻辑变量与输入逻辑变量之间一定的逻辑运算关系。这些内容在本教材的第2章已有一定阐述。计算机组成原理课程的相当大的一部分内容,涉及计算机各功能部件本身的逻辑功能与实现,以及计算机整机各部件之间的逻辑关系与连接。

3.2.2 字符类型数据的表示

字符是计算机中使用最多的信息形式之一,是人与计算机通信、交互作用的重要媒介。在计算机中,要为每个字符指定一个确定的编码,作为识别与使用这些字符的依据。这些编码的值,是用一定位数的基2码的两个基本符号1和0进行编码给出的。

1. ASCII码和EBCDIC码

使用得最多的、最普遍的是ASCII字符编码,即American Standard Code for Information Interchange,如表3.4所示。

表3.4 ASCII字符编码表

$b_3b_2b_1b_0$ \ $b_6b_5b_4$	000	001	010	011	100	101	110	111
0 0 0 0	NUL	DLE	SP	0	@	P	、	p
0 0 0 1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	STX	DC2	“	2	B	R	b	r
0 0 1 1	ETX	DC3	#	3	C	S	c	s

续表

$b_6b_5b_4$ / $b_3b_2b_1b_0$	000	001	010	011	100	101	110	111
0 1 0 0	EOT	DC4	$	4	D	T	d	t
0 1 0 1	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	ACK	SYN	&	6	F	V	f	v
0 1 1 1	BEL	ETB	‘	7	G	W	g	w
1 0 0 0	BS	CAN	(	8	H	X	h	x
1 0 0 1	HT	EM	)	9	I	Y	i	y
1 0 1 0	LF	SUB	*	:	J	Z	j	z
1 0 1 1	VT	ESC	+	;	K	[	k	{
1 1 0 0	FF	FS	,	<	L	\	l	|
1 1 0 1	CR	GS	—	=	M	]	m	}
1 1 1 0	SO	RS	.	>	N	↑	m	~
1 1 1 1	SI	US	/	?	O	-	o	DEL

从表中可以看到：每个字符是用7位基2码表示的，其排列次序为$b_6b_5b_4b_3b_2b_1b_0$，在表中的$b_6b_5b_4$为高位部分，$b_3b_2b_1b_0$为低位部分。而一个字符在计算机内实际上用8位表示。正常情况下，最高一位b_7为0。在需要奇偶校验时，这一位可用于存放奇偶校验位的值，此时称这一位为校验位。

ASCII是由128个字符组成的字符集。其中编码值0～31不对应任何可印刷(或称有字形)字符，通常称它们为控制字符，用于通信中的通信控制或对计算机设备的功能控制。编码值为32的是空格(或间隔)字符SP。编码值为127的是删除控制DEL码。其余的94个字符称为可印刷字符，有人把空格也计入可印刷字符时，则称有95个可印刷字符。请注意，这种字符编码中有如下两个规律。

(1) 字符0～9这10个数字符的高3位编码为011，低4位为000～1001。当去掉高3位的值时，低4位正好是二进制形式的0～9。这既满足正常的排序关系，又有利于完成ASCII码与二进制码数值之间的类型转换。

(2) 英文字母的编码值满足正常的字母排序关系，且大、小写英文字母编码的对应关系相当简便，差别仅表现在b_5一位的值为0或1，有利于大、小写字母之间的编码变换。

另有一种字符编码主要用在IBM计算机中的EBCDIC编码(Extended Binary Coded Decimal Interchange Code)。它采用8位码，有256个编码状态，但只选用其中一部分。0～9这10个数字符的高4位编码为1111，低4位仍为0000～1001。大、小写英文字母的编码同样满足正常的排序要求，而且有简单的对应关系，即同一个字母的大、小写的编码值仅最高的第二位的值不同，易于判别与变换。

2. 字符串的表示

随着计算机在文字处理与信息管理中的广泛应用,字符串已成为最常用的数据类型之一。许多计算机中都提供字符串操作功能,一些计算机还给出读写字符串的机器指令。

字符串是指连续的一串字符。通常方式下,它们占用主存中连续的多个字节,每个字节存一个字符。当主存字由2个或4个字节组成时,在同一个主存字中,既有按从低位字节向高位字节的顺序存放字符串内容的,也有按从高位字节向低位字节的次序顺序存放字符串内容的。这两种存放方式都是常用方式,不同的计算机可以选用其中任何一种。例如,IF　A>B　THEN　READ (C)就可以有两种不同的存放方式,如图3.2所示。

I	F		A
>	B		T
H	E	N	
R	E	A	D
(	C	)	

(a)

A		F	I
T		B	>
	N	E	H
D	A	E	R
	)	C	(

(b)

图3.2　字符串的两种存放方式

假定每个主存字由4个字节组成,图3.2(a)是按从高位字节向低位字符的次序存放字符串,图3.2(b) 按从低位字节向高位字节的次序存放字符串。主存中每个字节存的都是相应字符的ASCII编码值。例如对图3.2(a)来说,每个字节分别存放的是十进制的73、70、32、65、62、66、32、84、72、69、78、32、82、69、65、68、40、67和41和32。

3. 中文汉字的编码

西文字符是一种拼音文字,用30个左右的字母可以拼写出所有单词,再加上一些数学符号、标点符号等辅助字符,就可以构成一个完整的西文字符集,字符总数不超过256个,所以使用7位或8位二进制位就可以表示。我国是个多民族的国家,使用包括汉、蒙、藏、朝鲜、僮、苗、哈尼、维吾尔等近60种民族文字,其中用得最多且最广的是汉字。汉字也是字符,但有其特殊性,汉字是表意文字,汉字的数量很多,总数超过6万个。一个字就是一个方块图形,编码就要复杂一些,也给汉字在计算机内部的表示、汉字的传输与交换、汉字的输入与输出等带来了一系列问题。下面对汉字的编码、输入、存储和输出编码进行介绍。

1) 汉字内码

汉字内码又称机内码,是指用于汉字信息的存储、交换、检索等操作的机内代码,一般采用2个字节表示。各种不同输入法输入的汉字,其内码在计算机中是相同的。汉字内码等于汉字国标码加上8080H,即表示汉字的两个字节信息的最高一个二进制的值必定为1,例如"中"字的机内码为D6D0H,这有利于在文字处理软件中区分中文字符和西文字母。

计算机中的汉字编码都是通过软件定义和处理的,硬件不直接提供对汉字处理的支持。

1981年,我国制定了"中华人民共和国国家标准信息交换汉字编码",代号为GB 2312—1980。在这种编码的字符集中,一共收录了7445个汉字和图形,其中汉字6763个、图形682个。所谓区位码,是将上述国家标准局公布的6763个两级汉字分为94个区,每个区分94位,一个汉字所在的区号和位号简单地组合在一起就构成了这个汉字的区位码,其中高两位表示区号,低两位表示位号,都采用十进制数。实际上也就是把汉字表示成二维数组,每个汉字在数组中的下标就是区位码。区位码可以唯一地确定一个汉字或符号。例如"中"字位于54区48位,"中"字的区位码即为"5448"。区位码的值加上十六进制的2020成为国标码。

2000年公布的汉字编码的国家标准是GB 18030,该标准收录了27484个汉字。编码标准采用单字节、双字节(2B)、四字节(4B)。GB 18030是简体中文字符集GB 18030—2000

和GB 18030—2005的代号。GB 18030—2005的标准名称是“中华人民共和国国家标准GB 18030—2005：信息技术：中文编码字符集(Chinese National Standard GB 18030—2005：Chinese coded character set)”。GB 18030—2005取代GB2312、GB 1300、GBK、18030—2000和Big5标准，支持Unicode的CJK统一汉字，共收录70244个汉字。GB 18030编码在码位空间上与Unicode标准对应。

2）汉字输入码

键盘是计算机系统最重要的输入设备，用于西文这种小字符集的文字输入是很方便的。由于汉字是大字符集，若设计并使用专门的汉字输入键盘则有许多困难，例如键数太多、查找不方便、成本过高等。经济又便捷的输入方式还是选用普通的键盘，采用几个键的组合代表一个汉字，这就需要解决汉字的输入码问题。

汉字输入码也称外码，是为了将汉字输入计算机而编制的代码，是代表某一汉字的一串键盘符号。汉字输入码的种类有很多，主要可划分为以下几种。

(1）数字编码，如区位码、国标码、电报码等。

(2）拼音编码，如全拼码、双拼码、简拼码等。

(3）字形编码，如王码五笔、郑码、大众码等。

(4）音形编码，如表形码、钱码、智能ABC等。

在输入汉字的过程中，输入一个汉字需要敲击的键的次数、用到的规则、方便程度和输入的速度，不同的输入码方案会有所不同，可谓各有长短，不同的人也会有自己的偏爱。

在键盘输入法之外，还有手写汉字联机识别输入和印刷汉字扫描输入后自动识别两种方法，现均已达到实用水平。现在还有一种用语音输入汉字的方法，虽然简单易操作，但离实用阶段尚有差距。

3）汉字字形码

汉字字形码又称汉字字模码，是对汉字字形经过点阵数字化后形成的一串二进制数，用于汉字的显示和打印。每个汉字的点阵规模可以有所不同，通常有以下几种类型。

(1）简易型汉字：16×16，32字节/汉字。

(2）普通型汉字：24×24，72字节/汉字。

(3）提高型汉字：32×32，128字节/汉字。

将所有汉字的字模点阵代码按内码顺序集中起来，构成了汉字库。汉字库可以被保存在磁盘，固化在内存或使用时调入内存，固化在打印机的逻辑电路中。

3.2.3 多媒体信息编码

计算机能对图画、声音、音乐和电影等信息进行各种处理，如支持建筑/机械等图纸制作、图像扫描输入和处理、音乐合成、语音识别与合成、视频会议等。这势必涉及如何在计算机内部表示相关媒体信息的问题。

1. 图的编码

在计算机中，图可以分成图像(Image)和图形(Graphics)两种类型。通常可以用矢量图(Vector Graphics)表示，也可以用位图(Bitmap或Bitmapped Image)表示。

图像表示法类似于汉字的字模点阵码。只是汉字描述的仅仅是形状(即字模)，而对于图像除了要描述的形状外，还要描述其颜色或灰度。例如，用8位二进制数表示一个像素点

的灰度,用 3 个字节分别表示彩色图像一个像素点的 3 个彩色分量(R、G、B)的强度。

图形表示法是根据画面中所包含的内容,分别用几何要素(如点、线、面、体)和物体表面的材料与性质以及环境的光照条件、观察位置等来进行描述,如工程图纸、地图等可采用图形表示,它们易于加工处理,数据量也少。汉字字形的轮廓描述方法就属于图形表示。

2. 声音的编码

计算机处理的声音可以分为如下 3 种:第 1 种是语音,即人的说话声;第 2 种是音乐,即各种乐器演奏出的声音;第 3 种是效果声,如掌声、打雷、爆炸等声音。在计算机内部可以用波形法和合成法两种方法来表示声音。所有的声音都可用波形法来表示,但更多用于语音和效果声,对于音乐声,则用合成法来表示更好一些。

声音可以用一种连续的随时间变化的声波波形来表示。这种波形反映了声音在空气中的振动。计算机要能够对声音进行处理,必须将声波波形转换成二进制表示形式,这个转换过程称为声音的"数字化编码"。声音的数字化编码过程分为以下 3 步。

(1) 以固定的时间间隔对声音波形进行采样,使连续的声音波形变成一个个离散的采样信号(即样本值),每秒采样的次数被称为采样频率,通常采用 44.1kHz、22.05kHz 和 11.025kHz 这 3 种频率,也可以自行选择。采样频率越高,声音的保真度越好。

(2) 对得到的每个样本值进行模数转换(称为 A/D 转换),将每个样本值用一个二进制数字量来表示。这个过程即所谓的量化处理。转换后的二进制数的位数一般可以有两种选择:16 位或 8 位。位数越多,量化精度越高,噪声越小,声音质量也就越好。

(3) 对产生的二进制数据进行编码(有时还需进行数据压缩),以按照规定的统一格式进行表示。表 3.5 是波形法采用的几种不同参数的数字化声音信息的比较。

表 3.5 波形法采用的几种不同参数的数字化声音信息的比较

几种不同参数的数字化声音信息			
采样频率(kHz)	量化精度(bit)	每分钟的数据量(MB)	质量与应用
44.1	16	5.3	相当于激光唱片质量,用于最高质量要求的场合
22.05	16 8	2.65 1.32	相当于调频广播质量,可用作伴音和声响效果
11.025	16 8	1.32 0.66	相当于调幅广播质量,可用作伴音和解说词

声音的另一种表示方法是合成法。它主要适用于音乐在计算机内部的表示。它把音乐的乐谱、弹奏的乐器类型、击键力度等用符号进行记录。目前广为采用的一种标准为 MIDI (Musical Instrument Digital Interface)。例如,在 MIDI 标准中,一个 MIDI 文件(扩展名为 mid)中包含了一连串的 MIDI 消息。每个 MIDI 消息由若干字节组成,通常第一个字节为状态字节,其后为 1 个或 2 个数据字节。状态字节的特征是最高位为 1,它用来指出紧随其后的数据字节的用途和含义。数据字节的最高位为 0,后面是 MIDI 消息的信息内容。例如,表示一个"中央 C"的 MIDI 消息可由以下 3 个字节组成:90h、3Ch、40h。其中 90 是状态字节,它表示一个音符的开始;3C 表示击键的位置为"中央 C";40 表示击键的速度为中等。与波形表示方法相比,采用合成法的 MIDI 表示,其数据量要少得多(相差 2~3 个数量

级），编辑修改也比较容易。但它主要适用于表现各种乐器所演奏的乐曲，不能用来表示语音等其他声音。

为了处理上述两类数字声音信息，计算机内部有一个相应的声音处理硬件（如声卡），用来完成对各种声音输入设备输入的声音进行数字化编码处理，并将处理后的数字波形声音还原为模拟信号声音，经功率放大后输出。有的声音处理硬件可外接 MIDI 键盘，将弹奏的乐曲以 MIDI 形式输入计算机内，并将计算机处理后的 MIDI 乐曲经合成器（波表合成器或频率合成器）合成为音乐声音后输出。

3. 视频信息的编码

视频信息的信息量最丰富，它是一种最有感染力的承载信息媒体。视频信息的处理是多媒体技术的核心。计算机通过在内部安装一个视频获取设备（如视频卡），将各类视频源（如电视机、摄像机、VCD 机或放像机等）输入的视频信号进行相应的处理，转化为计算机内部可以表示的二进制数字信息。

视频获取设备将视频信号转换为计算机内部表示的二进制数字信息的过程被称为视频信息的"数字化"。视频信息的数字化过程比声音更复杂一些，它是以一幅幅彩色画面为单位进行的。每幅彩色画面有亮度（Y）和色差（U，V）3 个分量，对 Y、U、V 这 3 个分量需分别进行采样和量化，得到一幅数字图像。表 3.6 是几种常用数字视频的格式。

表 3.6　几种常用数字视频的格式

名　　称	分辨率	量化精度	数据量
CCIR601	720×576×25	8+4+4	124Mb/s
CIF	360×288×25	8+4+4	26Mb/s
QCIF	180×144×25	8+4+4	6.5Mb/s

从表 3.6 中可以看出，数字视频信息的数据量非常大。例如，一分钟 CCIR601 数字视频，其数据量约为 1Gb，这样大的数据量无论是存储、传输还是处理，都相当困难。要解决这个问题就必须对数字视频信息进行压缩编码处理。在获取数字视频的同时立即进行压缩编码的处理，称为实时压缩。有些视频获取设备具有实时压缩的功能。

3.2.4　数值类型数据的表示

数值类型数据是表示数量多少、数值大小的数据。它们有多种类型和不同的表示方法。

日常生活中，用得最多的是带正、负符号的十进制数字串形式的表示方法，例如 3.1416、−256 等。这种形式的数据难以在计算机内直接存储与运算，主要用于计算机的输入/输出操作，是人-机间交换数据的媒介。

在计算机内有时可以使用二-十进制编码，即用 4 位基 2 码编码一位十进制数，一个多位的十进制数被表示为这种编码的数串。4 位基 2 码组成 16 个状态，一位十进制数仅有 10 个状态，因此，怎样从 16 个状态中选用其中 10 个就有非常多的方案。下面还会详细讨论。

在计算机中最常用的方法，是用二进制码表示数据，包括整数、纯小数、实数（通称浮点数）。这有利于减少所用存储单元的数量，又便于实现算术运算。为了更有效地、方便地统一表示正数、负数和零，对二进制数又可以选用原码、反码、补码等多种编码方案表示。

数值数据的表示与编码方案,与计算机的设计、实现关系十分密切,且涉及一些基础理论与处理技术,有必要进行较为详细的讨论,放在3.3节专门讲解。

数值数据用于表示数量的大小。讨论数值数据时,经常用到数值范围和数据精度两个概念。数值范围是指一种类型的数据所能表示的最大值和最小值;数据精度,通常用实数所能给出的有效数字的位数表示。这两个概念是不同的。在计算机中,它们的值与用多少个二进制位表示某种类型的数据,以及怎么对这些位进行编码有关。

二进制数主要分成定点小数、整数与浮点数3类,还有用4位二进制表示一个十进制数位的压缩数字串。先简要说明这3类二进制数的一般表示,再详细讨论其具体编码形式。

1. 定点小数的表示方法

定点小数是指小数点准确固定在数据某个位置上的小数。从实用角度看,都把小数点固定在最高数据位的左边,小数点前边再设一位符号位。按此规则,任何一个小数都可以被写成

$$N = N_S N_{-1} N_{-2} \cdots N_{-M}$$

如果在计算机中用 $m+1$ 个二进制位表示上述小数,则可以用最高(最左)一个二进制位表示符号(如用0表示正号,则1就表示负号),而用后面的 m 个二进制位表示该小数的数值。小数点不用明确表示出来,因为它总是固定在符号位与最高数值位之间,已成定论。定点小数的取值范围很小,对用 $m+1$ 个二进制位的小数来说,其值的范围为

$$|N| \leqslant 1 - 2^{-m}$$

即小于1的纯小数,这对用户算题是十分不方便的,因为在算题前,必须把要用的数,通过合适的“比例因子”化成绝对值小于1的小数,并保证运算的中间和最终结果的绝对值也都小于1,在输出真正结果时,还要把计算的结果按相应比例加以扩大。

定点小数表示法主要用在早期的计算机中,它最节省硬件。随着计算机硬件成本的大幅度降低,现代的通用计算机都被设计成能处理与计算多种类型数值数据的计算机。我们主要通过定点小数讨论数值数据的不同编码方案,定点小数也被用来表示浮点数的尾数部分。

2. 整数的表示方法

整数表示的数据的最小单位为1,可认为它是小数点定在数值最低位右面的一种数据。

整数又被分为带符号和不带符号的两类。对带符号的整数来说,符号位被安排在最高位,任何一个带符号的整数都可以被写成

$$N = N_S N_n N_{n-1} \cdots N_2 N_1 N_0$$

对于用 $n+1$ 位二进制位表示的带符号的二进制整数,其值的范围为

$$|N| \leqslant 2^n - 1$$

对不带符号的整数来说,所有的 $n+1$ 个二进制位均被用于数值,此时数值的范围是

$$0 \leqslant N \leqslant 2^{n+1} - 1$$

即原来的符号位被解释为 2^n 的数值。

有时也用不带符号的整数表示另外一些内容,此时它不再被理解为数值的大小,而被看成一串二进制位的某种组合。

在很多计算机中,往往同时使用不同位数的几种整数,如用8位(称为字节)、16位(称为半字)、32位(称为字)或64位(称为双字)二进制来表示一个整数,它们占用的存储空间

和所表示的数值范围是不同的。

3. 浮点数的表示方法

浮点数是指小数点在数据中的位置可以左右移动的数据。它通常被表示成

$$N = M \times R^E$$

这里的 M(Mantissa)被称为浮点数的尾数;R(Radix)被称为阶码的基数;E(Exponent)被称为阶的阶码。计算机中一般规定 R 为 2、8 或 16,是一个确定的常数,不需要在浮点数中明确表示出来。因此,要表示浮点数,一是要给出尾数 M 的值,通常用定点小数形式来表示。它决定了浮点数的表示精度,即可以给出的有效数字的位数。二是要给出阶码,通常用整数形式表示。它指出的是小数点在数据中的位置,决定了浮点数的表示范围。浮点数也要有符号位。在计算机中,浮点数通常被表示成如图 3.3 所示的格式。

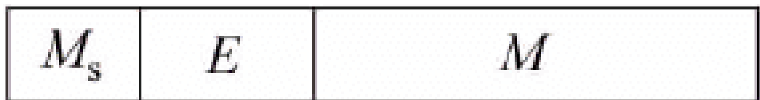


图 3.3 浮点数的格式

图 3.3 中 M_s是尾数的符号位,即浮点数的符号位,安排在最高一位;E 是阶码,紧跟在符号位之后,占用 m 位,含阶码的一位符号;M 是尾数,在低位部分,占用 n 位。

合理地选择 m 和 n 的值是十分重要的,以便在总长度为 $1+m+n$ 个二进制表示的浮点数中,既保证有足够大的数值范围,又保证有所要求的数值精度。例如,在 PDP-11/70 计算机中,用 32 位表示的一个浮点数,符号位占 1 位,阶码用 8 位,尾数用 23 位,数的表示范围约为$\pm 1.7\times 10^{\pm 38}$,精度约为十进制的 7 位有效数字。

若不对浮点数的表示格式作出明确规定,同一个浮点数的表示就不是唯一的。例如 0.5 也可以表示为 0.05×10^1、50×10^{-2}等。为了提高数据的表示精度,也为了便于浮点数之间的运算与比较,规定计算机内浮点数的尾数部分用纯小数形式给出,而且当尾数的值不为 0 时,其绝对值应大于或等于 0.5,这被称为浮点数的规格化表示。对不符合这一规定的浮点数,要通过修改阶码并同时左右移尾数的办法使其变成满足这一要求的表示形式,这种操作被称为浮点数的规格化处理,对浮点数的运算结果就经常需要进行规格化处理。

当一个浮点数的尾数为 0,不论它的阶码为何值,该浮点数的值都为 0。当阶码的值为它能表示的最小一个值或更小的值时,不管其尾数为何值,计算机都把该浮点数看成零值,通常称其为机器零,此时该浮点数的所有各位(包括阶码位和尾数位)都清为 0 值。

按国际电子电气工程师协会的 IEEE 标准,规定常用的浮点数的格式如表 3.7 所示。

表 3.7 IEEE 标准的浮点数格式

符号位	阶码	尾数	总位数	
短浮点数(单精度)	1	8	23	32
长浮点数(双精度)	1	11	52	64
临时浮点数	1	15	64	80

对短浮点数和长浮点数,当其尾数不为 0 值时,其最高一位必定为 1,在将这样的浮点数写入内存或磁盘时,不必保存该位,可左移一位尾数隐藏掉它,这种处理技术称为隐藏位技术。目的是用同样多位的尾数能多保存一个二进制位。为了保持浮点数的值不变,还要把原来的阶码值减 1。在将浮点数取回运算器执行运算时,再恢复该隐藏位的值和原来的阶码值。对临时浮点数(通常只出现在浮点运算器内部),不使用隐藏位技术。

从上述讨论可以看到,浮点数比定点小数和整数使用起来更方便。例如,可以用浮点数直接表示电子的质量 9×10^{-28} 克,太阳的质量 2×10^{33} 克,圆周率 3.1416 等。上述值都无法直接用定点小数或整数表示,要受数值范围和表示格式等各方面的限制。

4. 十进制数的编码与运算

十进制数的每一个数位的基为 10,但到了计算机内部,出于存储与计算方便的目的,必须采用基 2 码对每个十进制数位进行重编码,所需要的最少的基 2 码的位数为 $\log_2 10$,取整数为 4。4 位基 2 码有 16 种不同的组合,怎样从中选择出 10 个组合来表示十进制数位的 0~9,有非常多的可行方案,下面介绍其中的最常用的几种。

1) 十进制有权码

十进制有权码是指表示一个十进制数位的 4 位基 2 码的每一位有确定的位权。

用得最普遍的是 8421 码,即 4 个基 2 码位的权从高向低分别为 8、4、2 和 1,使用基 2 码的 0000、0001、…、1001 这 10 种组合,分别表示 0~9 这 10 个值。这种编码的优点是这 4 位基 2 码之间满足二进制的进位规则,而十进制数位之间则是十进制规则,故称这种编码为以二进制编码的十进制(Binary Coded Decimal)数,简称 BCD 码或二-十进制码。另一个优点是在数字符的 ASCII 码与这种编码之间的转换方便,即取每个数字符的 ASCII 码的低 4 位的值便直接得到该数字的 BCD 码,入/出操作简便。在计算机内实现 BCD 码之间的算术运算要复杂一些,在某些情况下,需要对加法运算的结果进行修正。修正规则如下。

(1) 若两个 8421 码每位数相加之和等于或小于 1001,即十进制的 9,不需要修正。例如,$(1)_{10}+(8)_{10}=(9)_{10}$ 的计算结果本身就是正确的。

(2) 若相加之和在 10 到 15 之间,一方面应向高位产生一进位,本位还要进行加 6 修正,进位是在进行加 6 修正时产生的。例如,$(4)_{10}+(9)_{10}=(\mathbf{1})_{10}(3)_{10}$ 就是如此。

(3) 若相加之和在 16 和 18 之间时,向高位的进位会在相加过程中自己产生,对本位还需进行加 6 修正。例如,$(7)_{10}+(9)_{10}=(\mathbf{1})_{10}(6)_{10}$ 就是如此。

另外几种有权码,如 2421、5211、84-2-1、4311 码(表 3.8),也都是用 4 位有权基 2 码表示一个十进制数位,但这 4 位基 2 码之间并不符合二进制规则。这几种有权码的特性如下。

(1) 当采用 2421、5211 和 4311 编码时,任何两个十进制数位相加产生 10 或大于 10 的结果,相应的基 2 码相加会向高一位产生进位,有利于实现逢十进位的计数和加法规则。

(2) 任何两个相加之和等于 9 的十进制数位的基 2 码,互为反码,即满足十进制数按 9 互补(9's Complement)的关系,有利于简化减法处理。表 3.8 给出的是上面提到的十进制数位的编码方案。

表 3.8　4 位有权码

十进制数	8421 码	2421 码	5211 码	84-2-1 码	4311 码
0	0000	0000	0000	0000	0000
1	0001	0001	0001	0111	0001
2	0010	0010	0011	0110	0011
3	0011	0011	0101	0101	0100
4	0100	0100	0111	0100	1000

续表

十进制数	8421 码	2421 码	5211 码	84-2-1 码	4311 码
5	0101	1011	1000	1011	0111
6	0110	1100	1010	1010	1011
7	0111	1101	1100	1001	1100
8	1000	1110	1110	1000	1110
9	1001	1111	1111	1111	1111

2）十进制无权码

十进制无权码是指表示一个十进制数位的 4 位基 2 码的每一位没有确定的位权。

在采用的无权码的一些方案中，早期用得比较多的是余 3 码（Excess-3 Code），是把原二进制的每个代码都加 0011 值得到的。它的主要优点是执行十进制数位相加时，能正确地产生进位信号，而且还给减法运算带来了方便，其编码结果在表 3.9 中给出。

格雷码是另外一种常用的二-十进制编码，是使任何两个相邻的代码只有一个二进制位的状态不同，其余 3 个二进制位必须有相同状态。这种编码方法的好处是，从一编码变到下一个相邻编码时，只有一位的状态发生变化，有利于得到更好的译码波形，在模拟→数字、数字→模拟转换的电路中得到更好的运行结果。用 4 个二进制位的格雷码表示十进制数的 10 个状态的方案很多。

表 3.9　4 位无权码

十进制数	余三码	格雷码(1)	格雷码(2)
0	0011	0000	0000
1	0100	0001	0100
2	0101	0011	0110
3	0110	0010	0010
4	0111	0110	1010
5	1000	1110	1011
6	1001	1010	0011
7	1010	1000	0001
8	1011	1100	1001
9	1100	0100	1000

3）数字串在计算机内的表示与存储

人们习惯使用十进制数，而在计算机内，采用二进制表示和处理数据更方便。因此，在计算机输入和输出数据时，要进行十进制和二进制之间的相互转换处理，这是多数应用环境中的实际情况。而在某些特定的应用领域中，如商业统计，其特点是运算简单而数据量很大，这样使输入输出过程中的进制转换所占的时间比例很大。从提高机器的运行效率考虑，也可以采用在计算机内部直接用十进制方式表示和处理数据，这要求计算机内部增加少量

硬件线路。目前,大多数通用性较强的计算机,都能直接处理十进制形式表示的数值。采用十进制表示数据的另一个目的是提高数据的表示范围和运算精度。就是说,十进制数在计算机内是以十进制的数位组成的数串形式存储与计算的,其位数,即串长是可变的,可规定最长可用位数,因此不受二进制整数和浮点数统一格式的约束。

十进制数串在计算机内主要有两种表示形式。

(1)**字符串形式**,即一个字节存放一个十进制的数位或符号位的字符。在主存中,这样的一个十进制数占用连续的多个字节,故为了指明这样一个数,需要给出该数在主存中的起始地址和位数(串的长度)。

对用这种方式表示的数据进行算术运算是很不方便的,因为每个数字符占用一个字节,其低4位的值表示数值,而高4位的值在进行算术运算时不具有数值的意义。因此,用这种方式表示的十进制字符串,主要用在非数值计算的有关应用领域中。

(2)**压缩的十进制数串形式**,即一个字节存放两个十进制的数位,它比前一种形式节省存储空间,又便于直接完成十进制数的算术运算,是广泛采用的较为理想的方法。

用压缩的十进制数串表示一个数,要占用主存连续的多个字节,每个数位占用半个字节(即4个二进制位),其值可用二-十进制编码(BCD码,数字符的ASCII码的低4位)表示,符号位也占用半个字节并存放在最低数字位之后,其值选用4位编码的6种冗余状态中的有关值,如用1100表示正号,用1101表示负号。在这种表示中,规定数值位加符号位之和必须为偶数,当其和不为偶数时,应在最高数字位之前补一个0。此时,表示一个数要占用该偶数值位的一半那么多个字节。例如,+123被表示成123C,-12被表示成012D。

要指明一个压缩的十进制数串,也需给出它在主存中的首地址和数字位个数(不含符号位),又称位长,位长为0的数其值为0。压缩的十进制数串表示方法的优点是位长可变,许多机器中规定该长度从0~31,有的甚至更长。

3.3 二进制数值数据的编码方案与运算算法

3.3.1 原码、反码、补码的定义

二进制数值数据包括二进制表示的定点小数、整数和浮点数。这里讲的编码方法,主要是如何能方便统一地表示正数、零和负数,并且尽可能地有利于简化对它们实现算术运算用到的规则。很容易想到,数据符号的正与负,可用一位二进制的0和1两个状态加以表示。数据的数值用多位二进制表示。最常用的编码方法有原码表示、补码表示和反码表示3种。为了讨论的方便,通常称表示一个数值数据的机内编码为机器数,而把它所代表的实际值称为机器数的真值。

1. 定点小数的编码方法

用定点小数引出数值的3种编码(原码、补码和反码)方案是最方便的。

1)原码表示法

原码表示法是用机器数的最高一位代表符号,以下各位给出数值绝对值的表示方法。其定义为

$$[X]_{原} = \begin{cases} X & 0 \leqslant X < 1 \\ 1 - X & -1 < X \leqslant 0 \end{cases} \tag{3.9}$$

例如，$X=+0.1011$，　$[X]_{原}=01011$

　　$X=-0.1011$，　$[X]_{原}=11011$

按定义，当 $X=-0.1011$ 时，$[X]_{原}=1-X=1.0000-(-0.1011)=11011$。这里的 X 为数的实际值，即相应机器数的真值，$[X]_{原}$ 为原码表示的机器数。

原码具有如下 3 个性质。

(1) 在原码表示中，机器数的最高位是符号位，0 代表正号，1 代表负号，以下各位是数的绝对值，即$[X]_{原}=$符号位$+|X|$。

(2) 在原码表示中，零有两种表示形式，即

$$[+0.0]_{原}=00000,\qquad [-0.0]_{原}=10000$$

设 X、Y 的真值分别为 $X=+0.0000$，$Y=-0.0000$。对正的 0.0000，按原码定义$[X]_{原}=00000$，对-0.0000，则有 $[Y]_{原}=1-Y=1+0.0000=10000$，因此零的原码有两种表示形式。

(3) 原码表示方法的优点是在数的真值和它的原码表示之间的对应关系简单，相互转换容易，用原码实现乘除运算的规则简单。缺点是用原码实现加减运算很不方便。要比较参与加减运算两个数的符号，要比较两个数的绝对值的大小，才能确定运算结果的数值和符号，因此在计算机中经常用后面介绍的补码实现加减运算。

2) 补码表示法

补码表示法是用机器数的最高一位代表符号，以下各位给出数值按 2 取模结果的表示方法，其定义为

$$[X]_{补}=\begin{cases}X & 0\leqslant X<1\\ 2+X & -1\leqslant X\leqslant 0 \quad \text{MOD}\quad 2\end{cases}\tag{3.10}$$

例如：

$$X=+0.1011,\qquad [X]_{补}=01011$$
$$X=-0.1011,\qquad [X]_{补}=10101$$

按补码的定义，当 $X=-0.1011$ 时，$[X]_{补}=2+X=10.0000+(-0.1011)=10101$。

补码具有如下 5 个性质。

(1) 在补码表示中，机器数的最高一位是符号位，0 代表正号，1 代表负号。机器数和它的真值的关系是$[X]_{补}=2\times$符号位$+X$。

(2) 在补码表示中，0 有唯一的编码，即$[+0.0]_{补}=[-0.0]_{补}=00000$。

假定 $X=+0.0000$，$Y=-0.0000$，依据补码定义，则有

$$[X]_{补}=X=00000,\quad [Y]_{补}=2+Y=10.0000+0.0000=10.0000=00000$$

此处最后一步实现按 2 取模，处在小数点左侧第二位上的 1 去掉了。

(3) 补码表示的两个数在进行加法运算时，可以把符号位与数值位同等处理，只要结果不超出机器能表示的数值范围，运算后的结果按 2 取模后，得到的新结果就是本次加法运算的结果，即机器数的符号位与数值位都是正确的补码表示，即

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}\quad \text{MOD}\quad 2\tag{3.11}$$

这一结论极为重要。

例如，$X=+0.1010$，$Y=-0.0101$，则$[X]_{补}=01010$，$[Y]_{补}=11011$，求得$[X]_{补}+[Y]_{补}=01010+11011=100101$，按 2 取模后，符号位左边一位上的 1 去掉，则最后结果为 00101，其真值为$+0.0101$。符号位与数值位均正确。

又如，$X_1=X_2=-0.1000$，则$[X_1]_补=[X_2]_补=11000$，那么$[X+X]_补=11000+11000=110000$，按2取模后得10000，它的真值为$-1$。由此看出，用补码表示定点小数时，它能表示$-1$的值。

(4) $[X]_补$与其真值的关系。假定$[X]_补=X_0X_1X_2\cdots X_n$，则有$[X]_补=2X_0+X$，此关系对$X$为正、为零和为负都是正确的。$X$为正时，$X_0$应为0，$[X]_补=2\times0+X=X$，$X$为负时，$X_0$应为1，$[X]_补=2\times1+X=2+X$，均与补码的定义吻合。由此又可以得到从$[X]_补$求$X$的关系，即由机器数求其代表的真值的关系如下：

$$\begin{aligned}X&=[X]_补-2X_0\\&=X_0X_1X_2\cdots X_n-2X_0\\&=-X_0+0.X_1X_2\cdots X_n\end{aligned}\tag{3.12}$$

当X为正数时，$X_0=0$，真值$X=[X]_补$；

当X为负数时，$X_0=1$，真值$X=-1+0.X_1X_2\cdots X_n=-(1-0.X_1X_2\cdots X_n)$。

例如$[X]_补=10110$，则X的真值$=-(1-0.0110)=-0.1010$。当已知数的补码需计算数的真值时可采用此法，此公式在推导补码乘法的运算算法中很有用。

(5) 补码数的算术移位。将$[X]_补$的符号位与数值位一起右移一位并保持原符号位的值不变，可实现除法功能(除以2)，即$[X/2]_补=X_0X_0X_1X_2\cdots X_{n-1}X_n$。今考虑$X$为正、负数两种情况。

设$X=0.0110$，$[X]_补=00110$，右移一位得00011，是X除以2的补码结果。

设$X=-0.0110$，$[X]_补=11010$，计算$[X/2]_补=11101$，再按式(3.12)求真值得到$X/2=-0.0011$，说明得到的确实是X除以2的结果。

为了得到一个数的补码表示，当然可以通过补码的定义求得，但更简便的办法如下。

① 当$X\geqslant0$时，$[X]_补$的符号位取0，数值位取X的各数值位上的值，此时有$[X]_补=[X]_原$。

② 当$X<0$时，$[X]_补$的符号位取1，将X的各数值位取反(0变1，1变0)再在最低位加1，以得到$[X]_补$的各数值位上的值。

从$[X]_原$求$[X]_补$时，对正数或零，有$[X]_补=[X]_原$；对负数，是符号位不变，各数值位变反后再在最低位执行加1操作。同理，由$[X]_补$求$[X]_原$时，对负数仍是符号位不变，各数值位变反后再在最低位执行加1操作。

在说明补码的性质(3)时，特别强调两个数的补码相加，仅在其运算结果不超出机器能表示的数值范围时，运算结果才是正确的，否则得到的结果不正确。如$[X]_补+[Y]_补=01001+01010=10011$，两个大于0.5的正数相加，结果的符号位变成负号，数值部分也是错误的。这是因为参加运算的两个数的和大于1，超出了机器所能表示的范围，产生了所谓的“溢出”。对负数也会产生溢出，如$[X]_补+[Y]_补=10101+10100=01001$，两个负数相加，结果的符号位却变成正号，说明结果是错误的。

为了方便判别结果是否溢出，某些机器采用变形补码，又称模4补码表示方法，其定义为

$$[X]_补=\begin{cases}X & 0\leqslant X<1\\4+X & -1\leqslant X\leqslant 0\quad \text{MOD}\quad 4\end{cases}\tag{3.13}$$

也就是常说的双符号位的补码表示。例如：

$$X=+0.1011,\qquad [X]_{补}=001011$$
$$X=-0.1011,\qquad [X]_{补}=110101$$

按模 4 补码定义，当 $X=-0.1011$ 时，$[X]_{补}=4+X=100.0000+(-0.1011)=110101$。从上式的结果可以看出，模 4 补码的表示就是在模 2 补码表示的符号位之前再增加与原符号同值的另一个符号位。

模 4 补码具有如下 2 个性质。

(1) 模 4 补码的两个符号位相同，00 表示正号，11 表示负号，其数值位与其模 2 补码相同。当符号位为 01 或 10 时，表示数值溢出。01 表示两个正数相加之和大于等于 1 的情况，通称数值"上溢"；为 10 时，表示两个负数相加，而其和小于 −1 的情况，通称数值"下溢"。最左面的 1 个符号位总是正确的符号位的值。

(2) 在模 4 补码表示中，零有唯一的编码，即 $[+0.0]_{补}=[-0.0]_{补}=000000$。模 4 补码能表示 −1，即为 110000，与模 2 补码的情况非常类似。

模 4 补码具有模 2 补码的全部优点，而且更容易检查加减运算中的溢出情况。有必要指出，存储每个模 4 的补码数时，只要存一个符号位，因为任何一个正确的数值，其模 4 补码的两个符号位总是相同的。只在把两个模 4 补码的数送往算术与逻辑运算部件 ALU 完成加减计算时，才把每个数的符号位的值同时送到 ALU 的两位符号位，即只在算术与逻辑运算部件中采用双符号位。

3) 反码表示法

反码表示法是用机器数的最高一位代表符号，数值位是对负数值各位取反的表示方法，其定义为

$$[X]_{反}=\begin{cases}X & 0\leqslant X<1\\(2-2^{-n})+X & -1<X\leqslant 0\quad \mathrm{MOD}(2-2^{-n})\end{cases}\qquad(3.14)$$

例如：

$$X=+0.1011,\qquad [X]_{反}=01011$$
$$X=-0.1011,\qquad [X]_{反}=10100$$

正数的反码与其原码、补码相同。

反码具有如下 3 个性质。

(1) 在反码表示中，机器数最高位为符号位，0 代表正号，1 代表负号。

(2) 在反码表示中，零有两个编码，即 $[+0.0]_{反}=00000$，$[-0.0]_{反}=11111$。

(3) 用反码进行两数相加时，若最高位有进位，还必须把该进位值加到结果的最低位，才能得到真正的结果，这一操作通称"循环进位"。

在现在的计算机系统中，反码较少被使用。

2. 整数的编码方法

与定点小数的 3 种编码方法类似，整数也可以用原码、补码和反码 3 种不同的编码方法表示。区别主要表现在以下两个方面。

(1) 定点小数的小数点位置严格地设置在数的符号位与最高数值位之间，因此，数的表示范围和编码的取模值与用多少位二进制表示一个数无关，该位数只影响数值的精度。

(2) 可以认为整数是小数点被设置在最低一位数值位的右边，机器数的最高位仍被用

作数的符号位。数值的表示范围,以及整数编码的取模值,都与表示一个数所用的二进制位数有关。

整数 3 种编码的定义、特性和相互间的变换方法,均与定点小数相应的 3 种表示类似。只是补码的取模值为 2^{k+1} 或 2^{k+2}(对变形补码),这里的 k 为二进制整数数值位的位数。例如:

$$X=+10101 \qquad [X]_{原}=[X]_{补}=[X]_{反}=010101$$
$$X=-10101 \qquad [X]_{原}=110101$$
$$[X]_{补}=101011$$
$$[X]_{反}=101010$$

以上两个数的变形补码分别为 **00**10101 和 **11**01011。这里的 k 均为 5。

3. 浮点数常用的编码方法

前面已经说到,在计算机内,浮点数被表示为如图 3.3 所示的格式。

通常情况下,浮点数的符号位 M_S 仍然采用 0 表示正号、1 表示负号的规则。数的尾数部分 M 采用定点小数形式表示,可用原码(或补码)等编码方式。讨论浮点数的编码方法的关键是确定对阶码部分的编码方法。

在多数通用计算机中,浮点数的阶码部分多采用整数形式的移码表示。对由 1 位符号位和 n 位数值位组成的二进制形式的阶码,其移码的定义为

$$[X]_{移}=2^n+X \qquad -2^{-n}\leqslant X<2^n \tag{3.15}$$

将这一定义与整数补码的定义相比较,有

$$[X]_{补}=\begin{cases}X & 0\leqslant X<2^n \\ 2^{n+1}+X & -2^n\leqslant X\leqslant 0 \quad \text{MOD} \quad 2^{n+1}\end{cases} \tag{3.16}$$

就可找出移码和补码之间的如下关系:

当 $0\leqslant X<2^n$ 时,$[X]_{移}=2^n+X=2^n+[X]_{补}$;

当 $-2^n\leqslant X<0$ 时,$[X]_{移}=2^n+X=(2^{n+1}+X)-2^n$。

这表明,由$[X]_{补}$得到 $[X]_{移}$的方法是变 $[X]_{补}$的符号为其反码。例如:

$$X=+1011,[X]_{补}=01011,[X]_{移}=11011$$
$$X=-1011,[X]_{补}=10101,[X]_{移}=00101$$

移码具有如下 2 个性质。

(1) 最高一位为符号位,但其取值与原码和补码都相反,1 代表正号,0 代表负号。

(2) 移码只用于表示浮点数的阶码,故只用于整数。

对移码一般只执行加减运算,在对两个浮点数进行乘除运算时,是尾数实现乘除运算,阶码执行加减运算。对移码执行加减运算时,需要对得到的结果加以修正,修正量为 2^n,即要对用移码求得的符号位取反后,得到的才是移码形式的正确结果。

在移码表示中,0 有唯一的编码,即 $[+0]_{移}=[-0]_{移}=1000\cdots0$。而且浮点数机器零的形式为 $000\cdots000$。当浮点数的阶码$\leqslant-2^n$时,不管尾数值大小如何,都属于浮点数下溢,被认为其值是 0。此时,移码表示的阶码值正好是每一位都为 0 的形式,这有利于简化机器中的判 0 线路。

到了 IEEE 的符点数标准 754 中,对上述内容做了某些补充与修正,见本教材第 4.3.1 节中的有关说明。

3.3.2 补码加、减运算规则和电路实现

在计算机中，通常总是用补码完成算术的加减法运算。其规则是

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}，[X-Y]_{补}=[X]_{补}+[-Y]_{补}$$

这表明，有了补码表示的被加(减)数和加(减)数，要完成计算补码表示的二数之和或二数之差，只需用二数的补码直接执行加减运算即可，符号位与数值位同等对待，一起参加运算，若运算结果不溢出，即不超出计算机所能表示的范围，则结果的符号位和数值位同时为正确值。此外，还可以看到，实现减运算时，用的仍是加法器线路，把减数的负值的补码送加法器即可。在有了一个数的补码之后，求这个数的负值的补码，是简单地把这个数的补码逐位取反再在最低位加 1 即可得到。例如，$[Y]_{补}=101101$，则$[-Y]_{补}=010011$，这大大简化了加减运算所用的线路和加减运算的实现方法。

下面的问题是如何检查加减运算中的溢出问题。通常有 3 种表述方式(说法)。

(1) 两个符号相同的补码数相加，如果和的符号与加数的符号相反，或两个符号相反的补码数相减，差的符号与减数的符号相同，都属于运算结果溢出。这种判别方法比较复杂，要区别加还是减两种不同运算情况，还要检查结果的符号与其中一个操作数的符号的同异，故很少使用。

(2) 两个补码数相加减时，若最高数值位向符号位送的进位值与符号位送向更高位的进位值不相同，也是运算结果溢出。

(3) 在采用双符号位(如定点小数的模 4 补码)运算时，若两个符号位的值不同(01 或 10)则是溢出。01 表明两个正数相加，结果大于机器所能表示的最大正数，称为“上溢”；10 表明两个负数相加，结果小于机器所能表示的最小负数，称为“下溢”；双符号位的高位符号位，不管结果溢出否，均是运算结果正确的符号值，这个结论在乘法运算过程中是有实际意义的。请注意，在采用双符号位的方案中，在寄存器和内存储器存储数据时，只需存一位符号，双符号位仅用在 ALU 的线路部分。

再次强调，这 3 种不同说法是对同一个事实的略有区别的表述。实现时用到的线路可以有所区别，但问题的实质是完全一样的。请看 $[X]_{补}+[Y]_{补}$的运算情况：

(1)	(2)	(3)	(4)	(5)	(6)
01011	10101	10100	10111	001011	110111
+01000	+11000	+11001	+10101	+001000	+110101
10011	101101	101101	101000	010011	1101100

这全都是溢出情况，前 4 个使用一个符号位，后 2 个使用两个符号位。用前面说的任何一种表述解释这里的溢出都是可以的。例如，对于(1)，从正加正得负，或数据位向符号位送的进位值为 1，而符号位送向更高位的进位值却为 0，二者不相同，或在(5)中使用双符号位方案时，其双符号位结果为 01，都是运算结果溢出。

凡补码加减运算其结果不属于上述情况的，均不是溢出，结果的符号位和数值位均正确。这里虽然讨论的都是加法运算，但对减运算亦适用。正减负等同正加正，正减正等同正加负，正如前面说过的，减运算也是用加法器完成的。例如：

```
  01011     11101      001011     111101
+00100    +11010     +000100    +111010
  01111     10111      001111     110111
  (1)       (2)        (3)        (4)
```

(1)和(2)使用一位符号位,(3)和(4)使用两位符号位,符号位送向更高位的进位值,不论其值为0或为1一律在取模后丢弃。

有了上述说明,就可以用图3.4的逻辑线路完成2个补码数的加减运算。

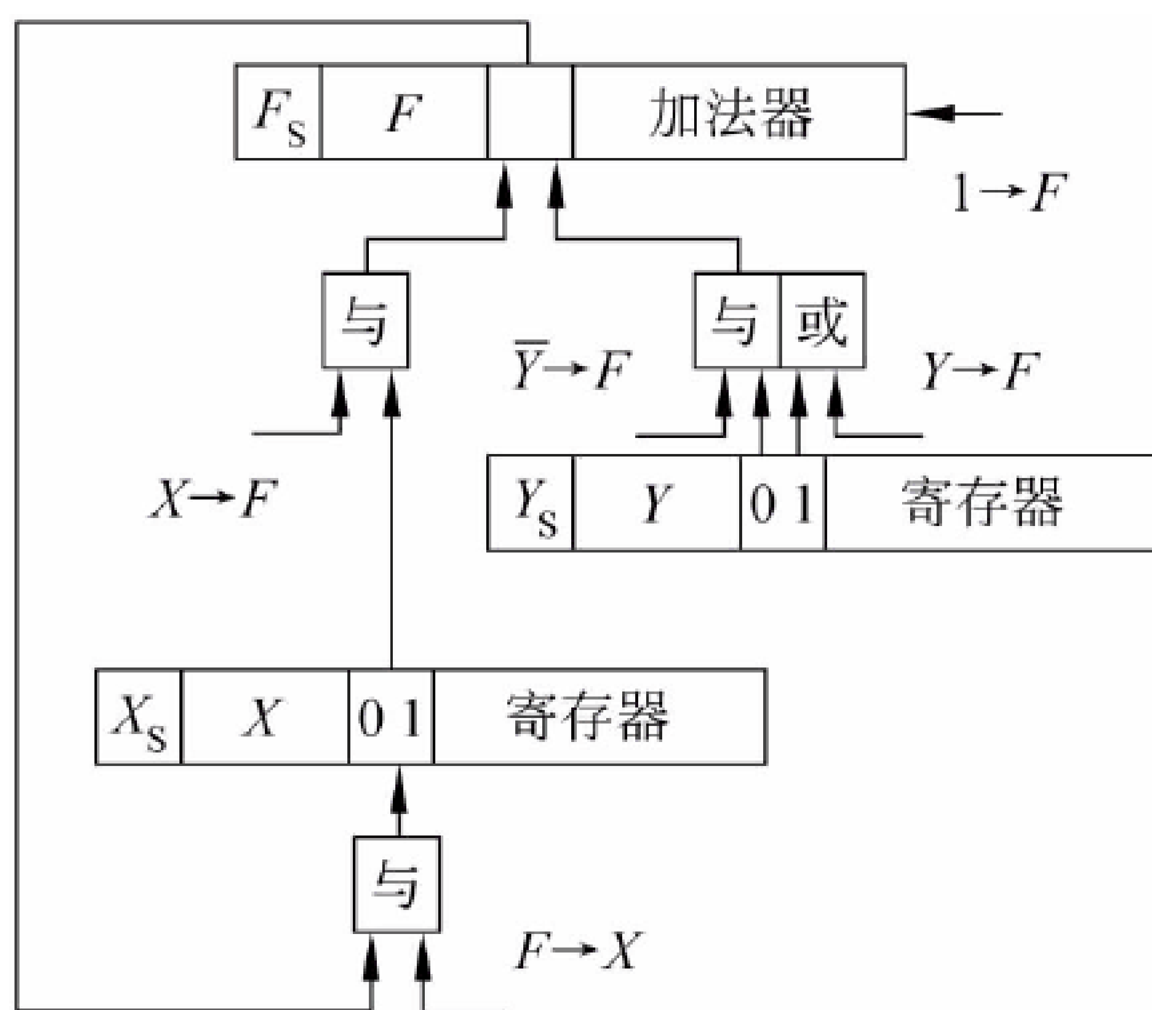


图3.4　实现补码加减运算的逻辑电路

运算前,X、Y寄存器分别存储被加(减)数和加(减)数,计算结果存回X寄存器;F为加法器,能在命令$X \to F$和$Y \to F$信号的控制下接收两个寄存器中的数据并完成加法运算,运算结果在$F \to X$命令信号的控制下接收回X寄存器中。为实现减运算,应将Y寄存器中补码数据的负数表示送到加法器F,这可以通过送Y寄存器中每位数据的反码并在F的最低位给出进位1输入信号变通完成,用$\bar{Y} \to F$和$1 \to F$两个控制信号实现控制。

3.3.3　原码一位乘法、除法的实现方案

1. 原码一位乘法的实现算法和电路实现

用原码实现乘法运算是十分方便的。原码表示的两个数相乘,其乘积的符号为相乘两数符号的异或值,数值则为两数绝对值之积。

假定

$$[X]_{原} = X_S X_1 X_2 \cdots X_n$$

$$[Y]_{原} = Y_S Y_1 Y_2 \cdots Y_n$$

则

$$[X \times Y]_{原} = [X]_{原} \times [Y]_{原} = (X_S \oplus Y_S)\ (X_1 X_2 \cdots X_n) \times (Y_1 Y_2 \cdots Y_n)$$

结果是把符号位和数值连接起来。

为了引出在计算机中实现定点原码一位乘法的具体方案,先看手工乘法运算的实际执行步骤。

假定:$X=0.1101$　$Y=0.1011$

```
      0.1 1 0 1
   ×  0.1 0 1 1
   ---------------
          1 1 0 1
        1 1 0 1
      0 0 0 0
    1 1 0 1
---------------
0.1 0 0 0 1 1 1 1
```

$X \times Y = 0.10001111$,符号为正

在手工计算时,其算法与执行步骤如下。

（1）依乘数每一位上的取值为1还是为0，决定相加数取被乘数的值还是取零值。

（2）各相加数从乘数的最低位求起，逐位变高并将相加数逐个左移一位，最后一步一次求和。

（3）符号位按正乘正、负乘负结果的符号位为正，正乘负、负乘正结果的符号为负的方案求出乘积的符号。

在计算机内实现原码乘法运算，则不能简单照搬上述方法，主要表现在以下诸方面。

首先，在运算器内很难实现多个数据同时相加，通常只能完成对两个数的求和操作。这一点比较容易解决，可以每求得一个相加数，就同时完成与上一次部分积的相加操作。

其次，在手工计算时，各相加数逐个左移一位，最终相加数的位数为相乘二数位数的两倍，而在计算机中，加法器的位数一般与寄存器的位数相同，而不是寄存器位数的两倍。这实际上也可以用另外的办法加以解决。手工计算时，各相加数是逐位左移一位，但很容易发现，在计算机内，在每次计算本次部分积之和时，前一次部分积的最低一位是不再参与相加计算的。这就意味着，若采用每求得一次部分积之后使其右移一位，则可以只用 N 位的加法器就能实现两个 N 位的数相乘，并有可能求得双倍位数的乘积。显而易见，若前一次部分积已经右移一位，就可以用其高位部分，再用加被乘数或加零的方法求得本次的部分积。

最后，手工计算时，乘数每一位的值是0还是1都能直接看见，而在计算机内，若采用放乘数的寄存器的每一位来直接决定本次相加数是被乘数还是零，实现起来是不方便的，若均采用该寄存器的最低一位来执行这种判别就简便多了。为此，可以在每求一次部分积，使放乘数的寄存器执行一次右移操作即可实现。若移位时，使其最高一位数值位接收加法器最低位的移位输出，则完成乘法运算后，该寄存器中保存的将是乘积的低位部分，而原来的乘数在逐位移位过程中已经丢失。

计算机内求乘积的符号，很容易用求相乘二数符号的半加和（异或值）实现。

有了上述说明，我们就可以得到如图3.5所示的实现原码一位乘法的逻辑电路框图。

乘法开始时，A 寄存器被清为零，作为初始部分积。被乘数放在 B 寄存器中，乘数放在 C 寄存器中。实现部分积和被乘数相加是通过给出 $A \to F$ 命令和 $B \to F$ 命令实现的，部分积右移一位送 A 寄存器是通过把 F 右斜一位的命令 $F/2 \to A$ 来控制完成的。

图3.5 实现原码一位乘法运算的逻辑线路框图

C 寄存器是用移位寄存器实现的，本身能在移位命令 $C/2 \to C$ 控制下自行移位，其最低位的值可用作 $B \to F$ 的控制命令。请注意，加法器最低一位的值，右移时将移入 C 寄存器的最高

数值位，使相乘之积的低位部分保存进 C 寄存器中，原来的乘数在逐位右移过程中丢失了。

图3.5中还给出了一个计数器 C_d，用来控制逐位相乘的次数。它的初值经常放乘数位数，以后每完成一位乘计算就使其逆向计数一次，待计数到0时，给出结束乘运算的控制信号。

图3.5中未画出求结果的符号的电路，可以通过对相乘两数的符号执行异或操作来完成。

计算机内实现原码乘法的具体过程如下。

假定　$X=0.1101$　　$Y=0.1011$

高位部分积	低位部分积/乘数	说明
00 0000	1 0 1 1	起始情况
+ 00 1101		乘数最低位为1，加 X
00 1101		
00 0110	1 1 0 1 1(丢失)	右移部分积和乘数
+ 00 1101		乘数最低位为1，加 X
01 0011		
00 1001	1 1 1 0 1(丢失)	右移部分积和乘数
+ 00 0000		乘数最低位为0，加0
00 1001		
00 0100	1 1 1 1 0(丢失)	右移部分积和乘数
+ 00 1101		乘数最低位为1，加 X
01 0001		
00 1000	1 1 1 1 1(丢失)	右移部分积和乘数

结果符号位为正，数值位为10001111。

这种乘法运算的控制流程如图3.6所示。

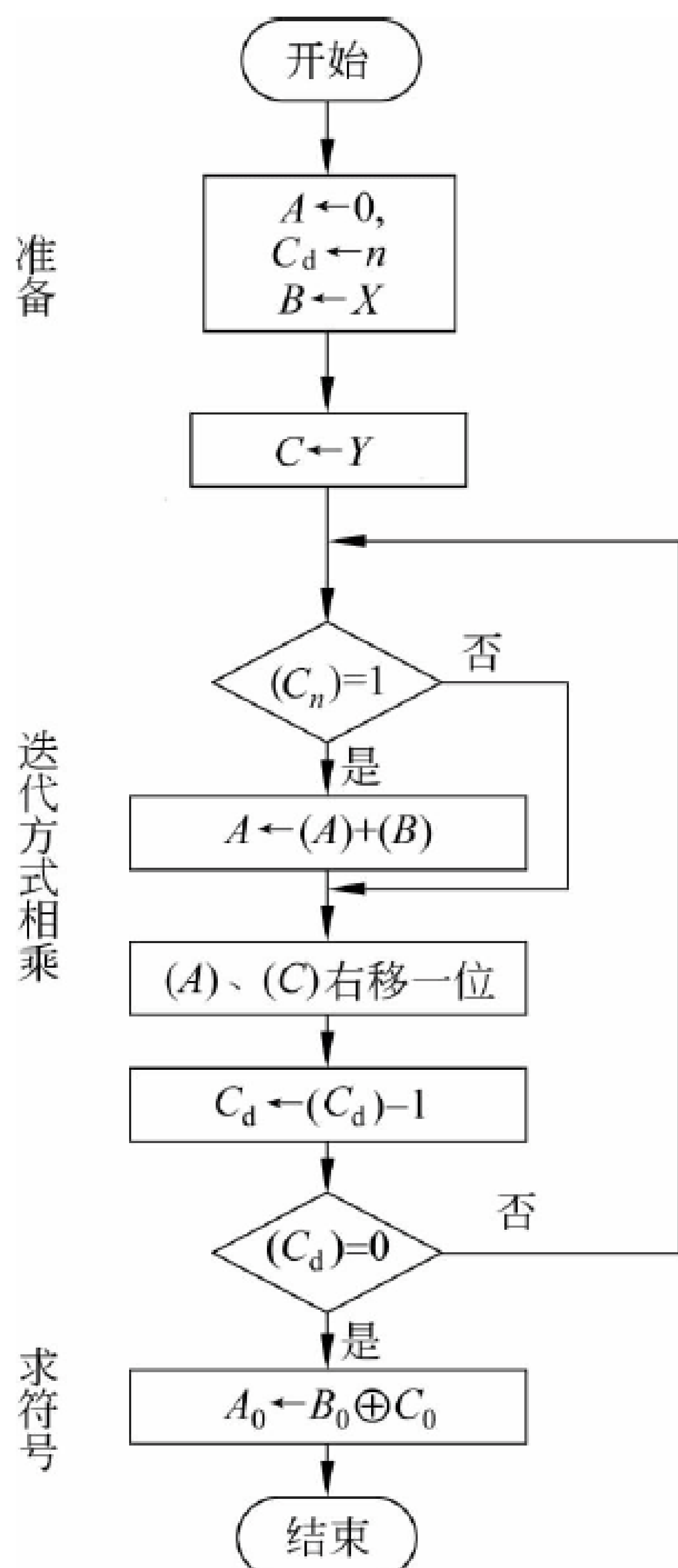


图3.6　原码一位乘运算流程

该图中，数据的位序号从左至右移按0、1、…、n 的次序编排，0位表示符号，共 n 位数值。

从流程图上可以清楚地看到，这里的原码一位乘是通过循环迭代的办法实现的，即按一定的时间顺序重复地使用最少量的硬件(寄存器、加法器、移位和传送门等)，把整个乘法过程变成为数据经过选通门和加法器实现相加、移位和寄存器接收的时序控制过程。即把 $X\times Y$ 可写为

$$P=2^{-1}\{2^{-1}[2^{-1}(2^{-1}\cdots(2^{-1}(0+Xy_n)+Xy_{n-1}+\cdots Xy_3)+Xy_2]+Xy_1\}$$

将上述公式展开，可写成

$$P_0=0$$
$$P_1=2^{-1}(P_0+Xy_n)$$
$$P_2=2^{-1}(P_1+Xy_{n-1})$$
$$\vdots$$
$$P_{i+1}=2^{-1}(P_i+Xy_{n-i})$$
$$\vdots$$
$$P_n=2^{-1}(P_{n-1}+Xy_1)$$

$$P \quad = P_n$$

此处的 P_0、P_1、…、P_{n-1} 称为部分积，P_n 为最终的乘积 P。

2. 原码一位除法的实现算法和电路实现

用原码实现定点除法运算是比较方便的，计算机中经常用原码的加减交替法完成除法运算。商的符号为相除二数符号的异或值，数值则为二数的绝对值之商。

就讲解除法的实现原理而言，恢复余数的除法方案更容易理解，并很容易由此推导出实现不恢复余数(即加减交替法)的除法算法。这种恢复余数的除法方案的缺点是明显的，当某一次减 Y 的差值为负时，要多一次 $+Y$ 恢复正余数的操作，降低了执行速度，又使控制线路变得复杂，因此在计算机中不被采用。在计算机中普遍采用的是不恢复余数的除法方案，它是对恢复余数除法的一种修正措施。

先看手工除法的计算过程。

假定　　$X=0.1011$，　$Y=0.1101$

```
              0.1101
       0.1101 ) 0.10110          X/Y = 0.1101，符号为正
                  1101           余数 = 0.0111×2^-4
                ------
                  10010
                   1101
                  ------
                   10100
                    1101
                   ------
                    0111
```

手工计算二进制除法的规则是判断被除数与除数绝对值的大小，若被除数小，则上商 0，并把被除数的下一位移下来(若存在)或在余数最低位补 0，再用余数和右移一位的除数比，若够除，则上商 1，否则上商 0。然后继续重复上述步骤，直到除尽(即余数为零)或已得到的商的位数满足要求为止。

与实现乘法类似的是，在计算机中，也不能全盘照搬手工计算除法的具体办法。主要问题是，手工计算的办法需要加法器的位数为除数位数的两倍，实现起来不合理。但仔细分析一下，会发现右移除数，可以通过左移被除数(余数)的方案替代，左移出界的被除数(余数)的高位都是无用的零，对运算不会产生任何影响。另外一个问题是，手工计算除法时，上商 0 还是 1 是计算者用观察比较的办法确定的，而在计算机中，只能用做减法后再判断结果的符号为负还是为正来确定。对恢复余数除法来说，当减出的差为负时，上商为 0，同时还应把除数再加到负差上去，恢复余数为原来的正值之后再将其左移一位。若减得的差为 0 或为正值时，就没有恢复余数的操作，上商为 1，余数左移一位。第 3 个问题是上商，手工除法中，求商是从高位向低位逐位求，而在计算机内，把求得的每一位商直接写进寄存器的不同的位是不可取的。通常，上商是通过把求得的每一位商上到存放商值的寄存器的最低一位，并把已求得的部分商左移一位。图 3.7 给出了实现原码一位除法运算的原理线路框图。

在不恢复余数的除法方案中，当某一次减得的差值为负时，不是恢复它为正差值后再继续运算，而是设法直接用这个负的差值直接求下一位商，其实现原理叙述如下。

(1) 在恢复余数的除法运算中，若第 $i-1$ 次求商时的余数为 $+R_{i-1}$，本次上商为 1，下一次求商用的办法是

$$R_i = 2R_{i-1} - Y$$

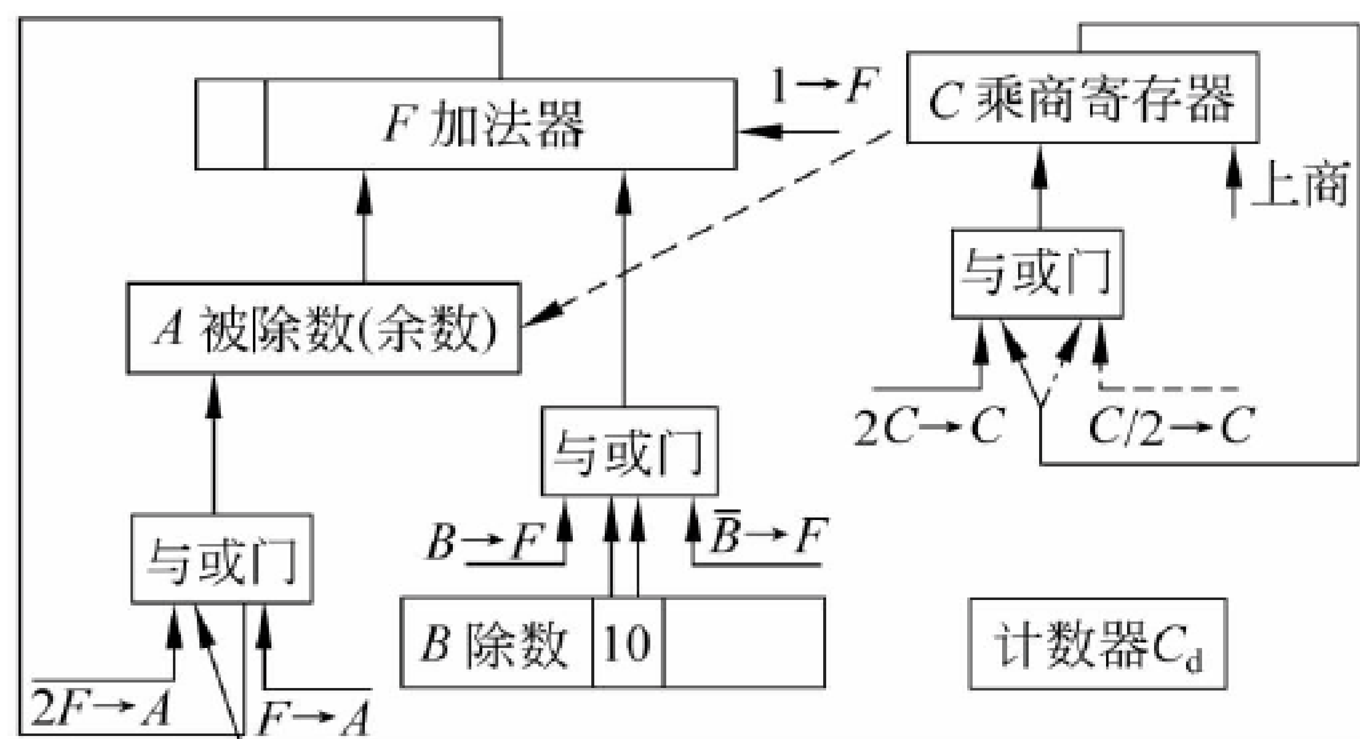


图 3.7　实现原码一位除法运算的逻辑线路框图

(2) 当 $R_i<0$ 时,第 i 位的商上 0,而恢复余数的操作结果应为 R_i+Y,下一次,即第 $i+1$ 次求商的减法操作是

$$R_{i+1}=2(R_i+Y)-Y=2R_i+2Y-Y=2R_i+Y$$

上述公式表明,当某一次求商,若减得的差值为负,即 $R<0$ 时,本次上商为 0,继续求下一位商时,可以不必恢复正余数,而是直接将负的差值左移一位(得 $2R$)后,再采用加上除数的办法来完成,即通过判别 $2R+Y$ 运算的结果为正还是为负来决定商的值。由此可得出不恢复余数的除法运算规则如下。

(1) 当余数为正时,商上 1,求下一位商的办法是正余数左移一位,用减去除数的办法得到。

(2) 余数为负时,商上 0,求下一位商的办法是负余数左移一位,用加上除数的办法得到。即在本方案中,在求得本位商值的同时,还要影响到用减去还是用加上除数的操作继续求下一位商,所以不恢复余数除法又叫加减交替除法。

下面给出不恢复余数除法执行除运算过程的一个例子。用的还是 $X=0.1011$,$Y=0.1101$ 这两个值,$[X]_补=00\ 1011$,$[Y]_补=00\ 1101$,$[-Y]_补=11\ 0011$。

被除数	商		操作说明
001011	0 0 0 0|0		开始情形
+ 110011		−Y	
111110	0 0 0 0|0		不够减,商上 0
111100	0 0 0|0 0		余数、商左移一位
+ 001101		+Y	
001001	0 0 0|0 1		够减,商上 1
010010	0 0|0 1 0		余数、商左移一位
+ 110011		−Y	
000101	0 0|0 1 1		够减,商上 1
001010	0|0 1 1 0		余数、商左移一位
+ 110011		−Y	
111101	0|0 1 1 0		不够减,商上 0
111010	|0 1 1 0 0		余数、商左移一位
+ 001101		+Y	
000111	|0 1 1 0 1		够减,商上 1

运算结果,商的数值位为 1101,符号位为相除二数符号的异或值 0,结果为+0.1101。

原码一位除的不恢复余数的运算过程的详细控制流程如图 3.8 所示。

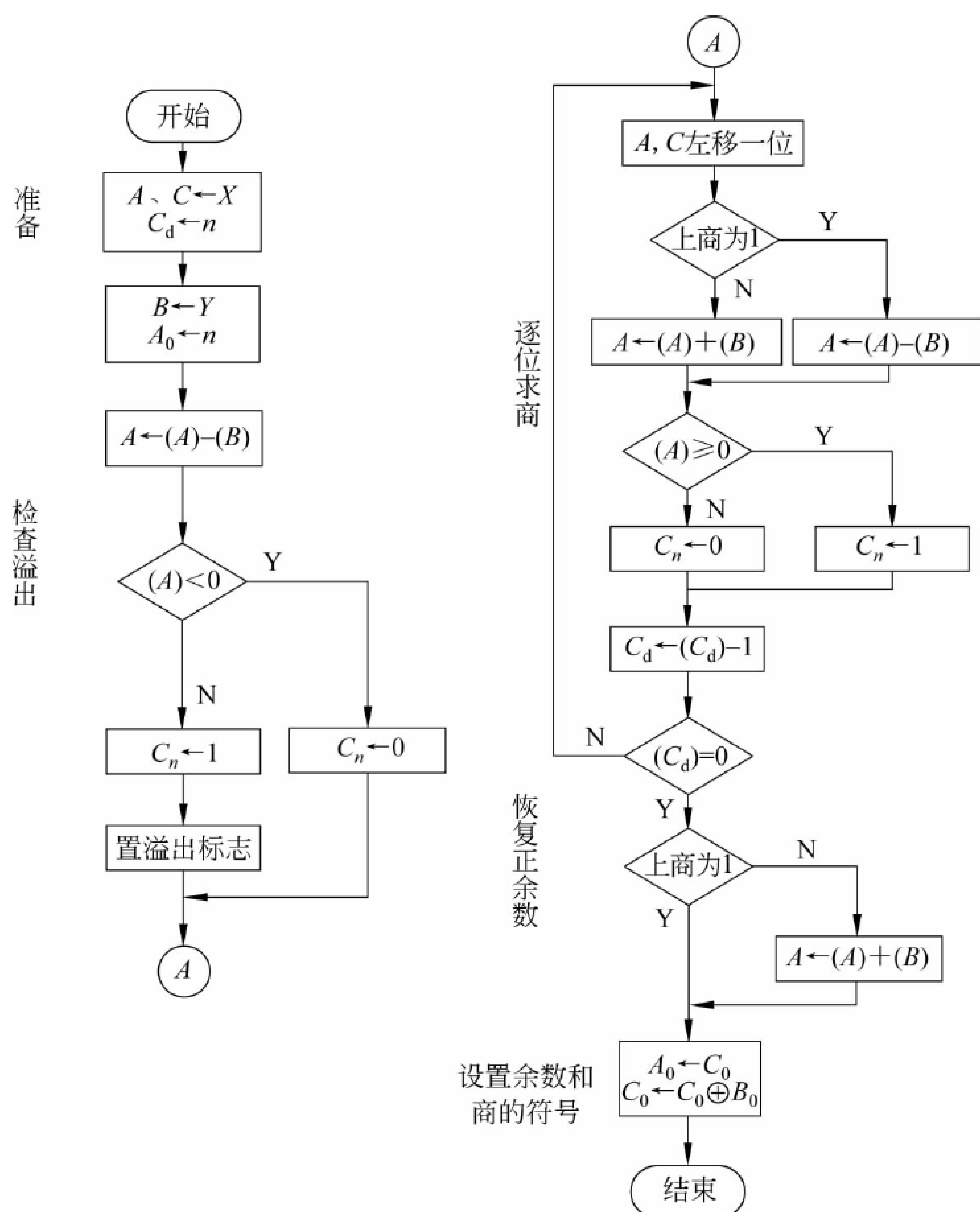


图 3.8　不恢复余数除法的运算的控制流程

至此，我们可把定点原码一位除的实现方案小结如下。

（1）除法运算首先需要检查出除数是否为 0，为 0 则是非法操作。

（2）对定点小数除法，首先要比较除数和被除数的绝对值的大小，需要防止出现数值溢出的错误。发现溢出，置溢出标记后，直接结束除运算的计算过程。

（3）商的符号为相除二数的符号的半加和。

（4）计算机中用加减交替法实现除法运算时，被除数的位数可以是除数的两倍，其低位的数值部分，开始时放在用于保存商的寄存器中。运算过程中，每求出一位商，放被除数和商的寄存器同时左移一位。

（5）在计算机中，求差和移位是在同一个操作步骤中完成的。

3.3.4　实现乘法、除法的其他方案

1. 定点补码一位乘的实现算法

在用原码实现乘法运算时，如相乘二数原来为补码表示（对加减运算更方便），则在相乘

之前,要还原负数的补码为它的原码形式;相乘之后,还要变负数积的原码为其补码形式,增加了操作步骤。为此,也有不少计算机直接用补码相乘,即用$[X]_补 \times [Y]_补$直接求$[X \times Y]_补$。这样做,对两个正数相乘没任何问题。当相乘的两个数中有一个为负数,或两个都为负数时,直接用它们的补码相乘,得到的积是否就是$[X \times Y]_补$呢?下面我们来分析一下这个问题。

若$[X]_补 = X_0X_1X_2\cdots X_n$,按前面讲过的,则在$[X/2]_补 = X_0X_0X_1X_2\cdots X_n$。同理,若已有$[X+Y]_补$,则$[(X+Y)/2]_补$也是将$[X+Y]_补$的符号连同其数值位同时右移一位,并使符号位保持不变,这对$[X+Y]_补$的符号为0(表示正)或为1(表示负)都是正确的。这表明,在讨论补码乘运算时,对被乘数或部分积的处理上与原码乘法有某些类似,差别仅表现在被乘数和部分积的符号位要和数值位一起参加运算。

若$[Y]_补 = Y_0Y_1Y_2\cdots Y_n$,则当$Y_0$为0时,$[Y]_补$与$[Y]_原$相同,很容易证明,直接用$[Y]_补$去乘$[X]_补$就能得到正确的$[X \times Y]_补$。但当$Y_0$为1时,即$Y$为负值时,则有

$$Y = -1 + \sum_{i=1}^{n} Y_i \times 2^{-i}$$

故有

$$X \times Y = X \times \left(-1 + \sum_{i=1}^{n} Y_i \times 2^{-i}\right) = X \times \sum_{i=1}^{n} Y_i \times 2^{-i} - X$$

这表明,当Y为负值时,用补码乘计算$[X \times Y]_补$,是用$[X]_补$乘上$[Y]_补$的数值位,而不用管$[Y]_补$符号位上的1,乘完之后,在所得的乘积中再减X,即加$[-X]_补$。就是说,当Y值为负时,需对求得的积再做一次加$[-X]_补$的校正操作,得到的才是补码形式的正确的积。这种方案需区分乘数的符号,按其正负做一步不同的操作。

实现补码乘法的另一个方案是比较法,是由BOOTH夫妇最早提出来的,故又称BOOTH法。这一方法的出发点是避免区分乘数符号的正负,而且让乘数的符号位也参加运算。可以用如下公式推导出它的实现原理:

$$\begin{aligned}
X \times Y &= X \times \left(-1 + \sum_{i=1}^{n} Y_i \times 2^{-i}\right) \quad \text{(逐项展开则得)} \\
&= X \times [-Y_0 + Y_1 \times 2^{-1} + Y_2 \times 2^{-2} + \cdots + Y_n \times 2^{-n}] \\
&= X \times [-Y_0 + (Y_1 - Y_1 \times 2^{-1}) + (Y_2 \times 2^{-1} - Y_2 \times 2^{-2}) \\
&\quad + \cdots + (Y_n \times 2^{-(n-1)} - Y_n \times 2^{-n})] \quad \text{(合并相同幂次项得)} \\
&= X \times [(Y_1 - Y_0) + (Y_2 - Y_1) \times 2^{-1} + \cdots + (Y_{n-1} - Y_n) \times 2^{-(n-1)} \\
&\quad + (0 - Y_n) \times 2^{-n}] \\
&= X \times \sum_{i=0}^{n} (Y_{i+1} - Y_i) \times 2^{-i} \quad \text{(写成累加求和的形式,得到实现补码乘运算的算法)}
\end{aligned}$$

由上述公式可以看到,比较法是用乘数中每相邻的两位判断如何求得每次的相加数。每两位Y_i和Y_{i+1}的取值有00、01、10和11这4种组合,则它们的差值分别为0、1、−1和0。依据上述公式可以看出,非最后一次的部分积,分别为上一次部分积的1/2(右移一位)的值R_j、$R_j+[X]_补$、$R_j-[X]_补$(即$R_j+[-X]_补$)和R_j,但一定要注意,最后一次求出的部分积即为最终乘积,不执行右移操作。

用此法计算乘积,需在乘数寄存器的低一位之后补充一位Y_{n+1},并使其初值为0,再增

加对 Y_n 和 Y_{n+1} 两位进行译码的线路，以区分出 $Y_{n+1}-Y_n$ 的 4 种不同的差值。对 n 位的数(不含符号位)相乘，要计算 $n+1$ 次部分积，并且不对最后一次部分积执行右移操作。此时的加法器最好采用双符号位方案。

2. 定点补码一位除法的实现方案

与补码乘法类似，也可以用补码直接完成除法运算，即用 $[X]_补/[Y]_补$ 直接求得 $[X/Y]_补$。补码除法的规则比原码除法的规则复杂一些。当除数和被除数用补表示时，判别是否够除，就不再是简单地用被除数(余数)减去除数，而是要比较它们的绝对值的大小。因此，若二数同符号，要用减法，若异号，则要用加法，请注意，这样求出来的商是反码形式的。

我们不准备对此进行更多的讨论，可以给出如下运算规则。

(1) 开始时，求第一位商，如果被除数与除数同号，用被除数减去除数，若二数异号，则用被除数加上除数的办法处理。

(2) 运算过程中确定商的值，若余数与除数同号，上商 1，左移一位后下次用余数减除数操作求商，若余数与除数异号，上商 0，左移一位后下次用余数加除数操作求商。

(3) 商的符号是在第一次求商试算时求出的，若定点除不溢出，得到的就是正确的符号位的值。

(4) 商的修正问题。在对精度要求不高时，将商的最低一位恒置 1。最大误差为$|2^{-n}|$。

若对商的精度要求较高，可对 n 位数求商 $n+1$ 次，按得到的不同结果对商进行修正。当商为负时，要在商的最低一位加 1，从反码的结果得到商的正确的补码值。

再求下去，可得下一位商并舍入；也可以不执行最后一步求商操作，而直接用在最低位上商 1 来结束除运算过程。

3. 双位乘法的实现方案

为了提高乘法的运算速度，也可以选用两位乘法的方案，即直接按乘数的每两位的取值情况，一次求出对应于该两位的部分积，此时只要增加少量的硬件电路，就可以使乘法的运算速度提高一倍，故被广泛地用在许多计算机中。

两位乘法运算的方案，既可以用来实现原码两位乘，也可以用来实现补码两位乘。

1) 原码两位乘

两位乘数的取值可以有 4 种可能组合，每种组合对应于以下操作：

(1) 00 相当于 $0\times X$，部分积 P_i 右移 2 位，不进行其他运算。

(2) 01 相当于 $1\times X$，部分积 P_i+X 后右称 2 位。

(3) 10 相当于 $2\times X$，部分积 P_i+2X 后右移 2 位。

(4) 11 相当于 $3\times X$，部分积 P_i+3X 后右移 2 位。

上面出现了 $+1X$，$+2X$，$+3X$ 这 3 种情况，$+X$ 容易实现，$+2X$ 可把 X 左移 1 位得 $2X$，在机器内通常采用向左斜 1 位传送来实现。可是 $+3X$ 一般不能一次完成，如分成两次进行，又降低了计算速度。解决问题的办法是：以 $+(4X-X)$ 来代替 $+3X$ 运算，在本次运算中只执行 $-X$，而 $+4X$ 则归并到下一步执行，因为下一步运算时，前一次的部分积已右移了两位，上一步欠下的 $+4X$ 在本步已变成 $+X$。实际线路中要用一个触发器 C 来记录是否欠下 $+4X$ 的操作尚未执行，若是，则 $1\rightarrow C$。因此实际操作要用 $Y_{i-1}Y_iC$ 这 3 位的组合值来控制乘法运算操作，运算规则如表 3.10 所示。

如果最后一次运算时欠下$+4X$,则部分积右移2位后还需补充完成$+X$操作。

表3.10　原码两位乘运算规则

Y_{i-1}	Y_i	C	操作		部分积右移
0	0	0	$+0$	$0\to C$	2位
0	0	1	$+X$	$0\to C$	2位
0	1	0	$+X$	$0\to C$	2位
0	1	1	$+2X$	$0\to C$	2位
1	0	0	$+2X$	$0\to C$	2位
1	0	1	$-X$	$1\to C$	2位
1	1	0	$-X$	$1\to C$	2位
1	1	1	$+0$	$1\to C$	2位

2）补码两位乘

我们可以方便地从上一节讲的用比较法实现补码一位乘的方案推导出补码两位乘的实现原理。当我们把补码两位乘理解为：合并原来两步补码一位乘为单步操作,则可找出如下对应关系。

假定上步乘法的部分积为$[P_i]_补$,本步的部分积应为

$$[P_{i+1}]_补=2^{-1}\{[P_i]_补+(Y_{n+1-i}-Y_{n-i})\times[X]_补\}$$

此后,下一步的部分积应为

$$[P_{i+2}]_补=2^{-1}\{[P_{i+1}]_补+(Y_{n-i}-Y_{n-1-i})\times[X]_补\}$$

将第一个公式中的$[P_{i+1}]_补$代入第二个公式中时,则得到

$$\begin{aligned}[P_{i+2}]_补 &= 2^{-1}\{2^{-1}\{[P_i]_补+(Y_{n+1-i}-Y_{n-i})\times[X]_补\}+(Y_{n-i}-Y_{n-1-i})\times[X]_补\}\\ &= 2^{-2}\{[P_i]_补+[(Y_{n+1-i}-Y_{n-i})+2\times(Y_{n-i}-Y_{n-1-i})]\times[X]_补\}\\ &= 2^{-2}\{[P_i]_补+[(Y_{n+1-i}+Y_{n-i}+2\times Y_{n-1-i})]\times[X]_补\} \qquad (3.17)\end{aligned}$$

式(3.17)表明,在有了部分积$[P_i]_补$之后,再求部分积$[P_{i+2}]_补$,可用$[P_i]_补$加上乘数寄存器最低两位与附加位3位值的组合结果与被乘数$[X]_补$之积、再右移两位得到。

其中3位值的组合关系为

$$Y_{n+1-i}+Y_{n-i}+2\times Y_{n-1-i}$$

代入它们的8组取值,则得到如表3.11所示的结果。

表3.11　补码两位乘的组合结果

Y_{n-1-i}	Y_{n-i}	Y_{n+1-i}	组合值	$[P_{i+2}]_补$
0	0	0	0	$[P_i]_补/4$
0	0	1	1	$([P_i]_补+[X]_补)/4$
0	1	0	1	$([P_i]_补+[X]_补)/4$
0	1	1	2	$([P_i]_补+2[X]_补)/4$
1	0	0	-2	$([P_i]_补+2[-X]_补)/4$

续表

Y_{n-1-i}	Y_{n-i}	Y_{n+1-i}	组合值	$[P_{i+2}]_补$
1	0	1	−1	$([P_i]_补+[-X]_补)/4$
1	1	0	−1	$([P_i]_补+[-X]_补)/4$
1	1	1	0	$[P_i]_补/4$

表 3.11 结果表明，执行补码两位乘的过程中，有部分积$+[X]_补$、部分积$+[-X]_补$、部分积$+2[X]_补$与部分积$+2[-X]_补$这 4 种操作。除需要有把$[X]_补$、$[-X]_补$送加法器的线路外，还需要有把$[X]_补$、$[-X]_补$左斜一位送加法器的线路。与此相应的，加法器应使用 3 位符号位，以避免$[X]_补$左斜一位送加法器时运算结果溢出的情形。最后一点是，部分积和乘数每次应右移两位，运算器中应有支持右移两位的逻辑电路。

请注意：求部分积的次数和右移操作的控制问题。当乘数由 1 位符号和 15 位数据位组成时，求部分积的次数为$(1+15)/2$，即 8 次，而且最后一次的右移操作只右移一位。若数值位本身为偶数 n，则必须再增加一位符号位，使总位数仍为偶数，此时求部分积的次数为 $n/2+1$，而且最后一次不再执行右移操作。

4. 阵列乘法器

在 3.3.3 节介绍了手工乘法的执行过程。原则上，在计算机内也可以照搬这种算法，即用图 3.9 所示的一个阵列乘法器完成 $X \times Y$ 乘法运算($X=X_1X_2X_3X_4$，$Y=Y_1Y_2Y_3Y_4$)。为节省线路，可能需要把多位乘数划分成一组，用于乘运算的一个步骤，则整个乘法只用几步即可完成。

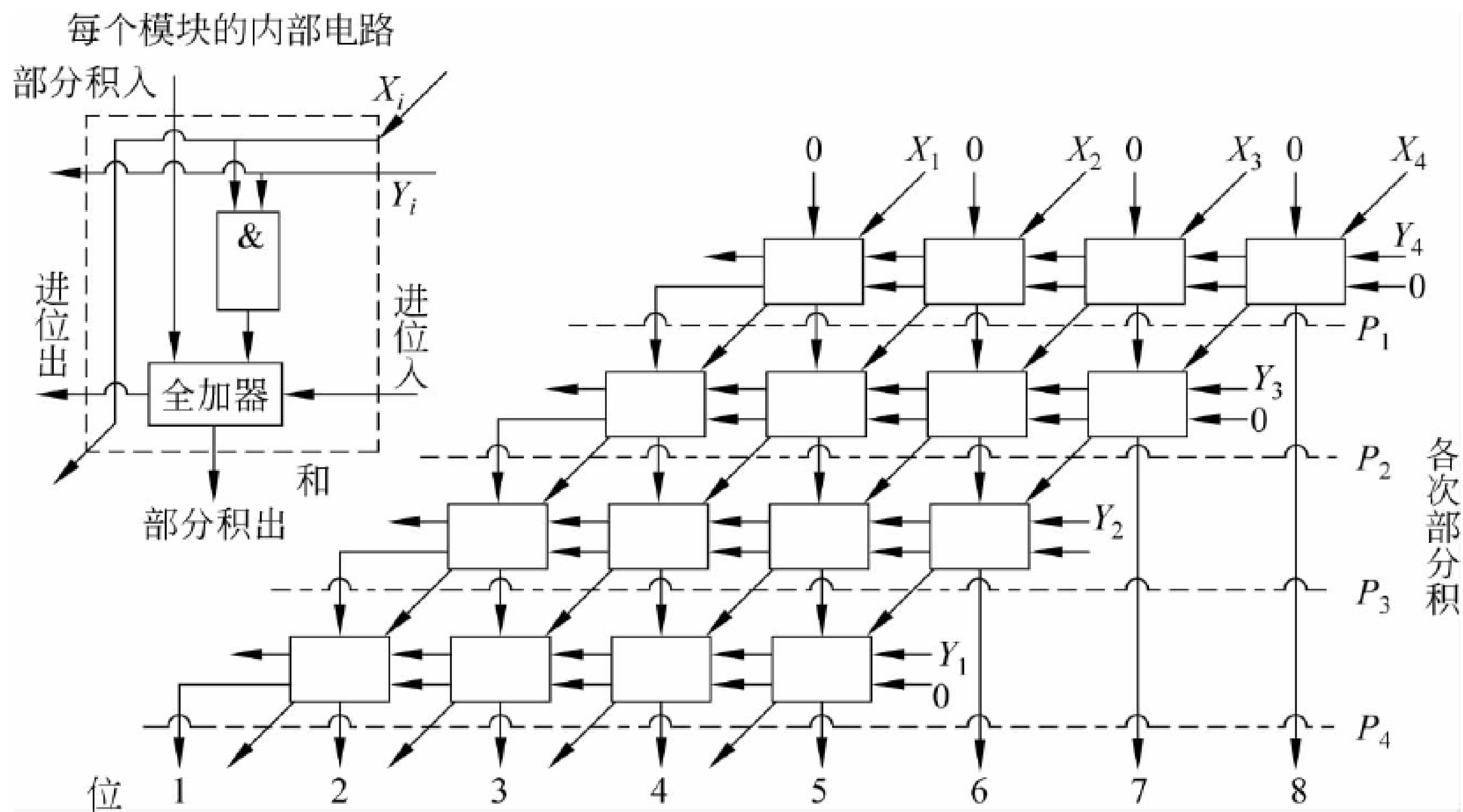


图 3.9 实现定点绝对值相乘的阵列乘法器

阵列的每一行由乘数 Y 的每一位数位控制，而各行错开形成的每一斜列则由被乘数的每一数位控制。图 3.9 中每一个方框都是一个细胞模块(Cellular Module)，包括一个与门和一位全加器。该方案所用加法器数量很多，但内部结构规则性强，标准化程度高，适于用超大规模集成电路的批量生产。

5. 用快速乘法器实现快速除法运算

按下式完成 M/D：

$$\frac{M}{D}=\frac{M\times F_0\times F_1\times F_2\times\cdots\times F_r}{D\times F_0\times F_1\times F_2\times\cdots\times F_r}$$

式中 $F_i(0\leqslant i\leqslant r)$为各次递乘因子，递乘几次后，可以使 $D\times F_0\times F_1\times F_2\times\cdots\times F_r\to 1$，则商应为 $M\times F_0\times F_1\times F_2\times\cdots\times F_r$。

若 M 和 D 为规格化正的二进制小数代码时，可写成

$$D=1-\delta\quad(0<\delta\leqslant 1/2)$$

那么可取 F_i的值如下：

$$\begin{aligned}F_0&=1+\delta\\D_0&=D\times F_0=(1-\delta)(1+\delta)=1-\delta^2\\F_1&=1+\delta^2\\D_1&=(1-\delta^2)(1+\delta^2)=1-\delta^4\end{aligned}$$

可见，当 i 增加时，D_i将很快趋近于 1，其误差为 $\delta^{2^{i+1}}$。

实际上求得 F_i的过程很简单，它应为$-D_i$的补码，即 $2-D_i(0\leqslant i\leqslant r)$。

则可以得到 $Q\approx 0.1100$，若算得更精确些，可以采用双倍字长乘运算，误差更小，求得的商为 $Q=1011$。

本章内容小结和学习方法建议

本章提供计算机系统能够处理的常用的数据类型及其表示的基本知识，也包括运算器部件执行数据运算可行计算方法的数学基础。这些内容与布尔代数、数字电路有一定的联系，和计算机组成与设计也直接相关。认识到这一点很重要，可以使自己的学习目标更清楚，明确为什么需要学习这部分知识，这些知识将用到哪些地方，怎么应用才能更好地学以致用，提升自己学习的主动性和学习效率。

本章的教学内容是以数字化信息编码的概念、二进制编码和不同码制之间的数据转换为基础，介绍了计算机中最常用的基本数据类型及其表示，这些内容是学习、理解和设计计算机硬件系统必须掌握的基础知识，是本章重点教学内容之一。

本章中另外一项重点教学内容是数值型数据的表示、编码和运算方法，包括定点小数(主要应用在浮点数的尾数部分)、整数(含带符号的和无符号的 2 种类型)和浮点数(结合浮点数 IEEE-754 标准的有关规定)。这里尤其要理解为什么要引入原、反、补码表示及其完成运算用到的方法，定点小数、整数补码加减法的运算规则，原码一位乘除法的原理性算法，以及如何判断运算结果是否溢出，这些是完成教学实验、设计实现简单 CPU 必定用到的基本知识。对算术运算用到的原理性逻辑线路、补码乘除法、加速乘除运算的可行方案一般了解即可。

针对提高计算机硬件系统可靠性的需要，结合二进制编码的概念，本章中简单介绍了检错纠错码知识及其应用，要求准确理解通过增大信息编码中的最小码距这一措施来实现检错纠错的道理，至于具体的编码过程和实现中所用到的电路一般了解即可。检错纠错是提高计算机可靠性、可用性的重要手段，作为容错的一种有效措施，在内存读写、外存设备读

写、数据传送（包括计算机网络）的过程中得到普遍应用，但作为计算机组成原理课程的教学内容只能是点到为止，不可能更详细地讲解。

学生在学习计算机组成原理课程的整个过程中，应该有意识地把各部分教学内容的内在联系梳理清楚，即现在学习的内容是建立在哪些已经学习过的知识的基础上，又会成为此后将要学习的哪些相关章节的必要准备，这是一个不断总结、逐步提高的认识过程，切忌把各章节的内容孤立起来，切勿使装入脑子中的新知识、新概念和相关技术变成一些孤立离散的内容。认真完成作业是学懂本章内容的主要方式之一。

习题与思考题

1. 把下面给出的几个十进制的数转化成二进制的数（无法精确表示时，小数点后取3位）、八进制的数和十六进制的数。

7+3/4，−23/64，27.5，−125.9375，5.43

2. 把下面给出的几种不同进制（以下标形式给出在右括号之后）的数转化成十进制的数。

$(1010.0101)_2$，$-(101101111.101)_2$，$(23.47)_8$，$-(1A3C.D)_{16}$

3. 完成下面几个二进制的数的算术运算。

1010.111+0101.101，1010.111−0101.101，1110×0101，10111101/1101

4. 回答奇偶校验码的用途是什么？写出下面2个二进制数的奇、偶校验码的值。

01010111，11010100

5. 设计完成对8位数据进行偶校验编码的逻辑线路和用于检查已得到的一个偶校验码字的合法性的逻辑电路。

6. 汉明校验码具有怎样的检错纠错能力？为实现对8位数据的汉明校验，应安排几个校验位？设计该汉明校验码的编码逻辑表达式、译码逻辑表达式。

7. 用第6题完成的设计，写出对应两个8位数据01010111和00000000的两个汉明码码字的值；若第一个码字中的最低一个数据位的值由1错变为0，第二个码字中的最低、最高的两个数据位由0错变成1，译码后的几个S的值都是什么？

8. 为什么在二-十进制编码（BCD码）中会出现多种编码方案？何为有权码系统？何为无权码系统？

9. 循环码的特点是什么？循环码用于数值计算方便吗？循环码主要用于什么场合？证明循环码为无权码系统。

10. 写出下面两组数的原、反、补码表示，并用补码计算每组数的和、差。双符号位的作用是什么？它只出现在什么电路之处？

(1) $X=0.1101$ $Y=-0.0111$

(2) $X=10111101$ $Y=-00101011$

11. 一个C语言程序在一台32位机器上运行。程序中定义了3个变量x、y和z，其中x和z为int型，y为short型。当$x=127$，$y=-9$时，执行赋值语句$z=x+y$后，x、y和z的值分别是________。

A. x=0000007FH，y=FFF9H，z=00000076H

B. x=0000007FH, y=FFF9H, z=FFFF0076H

C. x=0000007FH, y=FFF7H, z=FFFF0076H

D. x=0000007FH, y=FFF7H, z=00000076H

12. 浮点数加、减运算过程一般包括对阶、尾数运算、规格化、输入和判溢出等步骤。设浮点数的阶码和尾数均用补码表示,且位数分别为5位和7位(均含2位符号位)。若有两个数 $X=2^7\times29/32$,$Y=2^5\times5/8$,则用浮点加法计算 $X+Y$ 的最终结果是________。

A. 00111　1100010　　B. 00111　0100010

C. 01000　0010001　　D. 发生溢出

13. 假定有4个整数用8位补码分别表示为 r_1=FEH,r_2=F2H,r_3=90H,r_4=F8H,若将运算结果存放在一个8位寄存器中,则下列运算会发生溢出的是________。

A. $r_1\times r_2$　　B. $r_2\times r_3$　　C. $r_1\times r_3$　　D. $r_2\times r_4$

14. 仿照计算机的计算过程,用原码一位乘计算在10题的第(1)小题给出的两个定点小数的乘积。

15. 仿照计算机的计算过程,用原码一位除计算在10题的第(1)小题给出的两个小数 Y/X 的商和正确的余数。

第 4 章 运算器部件

运算器分为定点运算器和浮点运算器两种类型。运算器部件是计算机系统中的执行部件，分担对二进制数据进行各种算术、逻辑运算，也是计算机中央处理器 CPU 内部数据传送的重要通路。通常由执行数据的算术逻辑运算功能的线路 ALU、暂存被运算数据和中间结果的寄存器组(Regs)、保存标志位信息的标志寄存器 Flag(可能设置在控制器芯片中，但逻辑上把它理解为运算器部件的一个组成部分更好理解一些)三部分核心电路组成，当然还会用到其他一些辅助电路。本章重点介绍运算器的核心部件——算术逻辑运算单元 ALU、寄存器组的组成与工作原理。

在本章将给出 3 个不同特点的运算器实例。

第一个实例是作者设计的一个原理性的 8 位运算器模型，重点是复习数字电路和逻辑设计的基础知识，讲解简单运算器的功能设计及其工程实现，会用到 ABEL-HDL 硬件描述语言和现场可编程的 CPLD 芯片的基本知识。

第二个实例是 20 世纪 70 年代初期用在 CISC 结构的计算机中的运算器芯片 Am2901，将比较详细地介绍它的组成、工作原理及使用方法，并用它来构建教学实验设备中的运算器部件，要求学生重点关注。

第三个实例是 20 世纪 80 年代中期使用在 RISC 结构的 MIPS32 计算机的运算器部件，将对它的组成与功能进行简要概述，并对比它与 Am2901 运算器在组成、功能、执行运算的步骤等多个方面的区别。

4.1 算术逻辑运算单元的功能设计与线路实现

算术与逻辑运算线路是计算机运算器部件的核心电路，从功能的角度看，它要完成对数值数据的算术运算功能，给出运算结果的数值和结果的特征信息(例如结果的符号、向更高位的进位、结果是否为 0、结果是否溢出)，还要完成对逻辑数据的逻辑运算功能。本节将介绍设计实现 ALU(Arithmetic and Logic Unit)的方法和过程。

设计实现一位加法器的过程是：①写出加法器的真值表；②用布尔代数写出逻辑表达式；③对得到的逻辑表达式进行适当的化简，就可找出所用电路及其连接关系。每一位加法器实现对 2 个二进制数(X_n、Y_n)和一个进位输入(C_n)的加法运算，产生一位的相加之和(S_n)，以及一位的进位输出(C_{n+1})，图 4.1 中给出了这一设计结果。

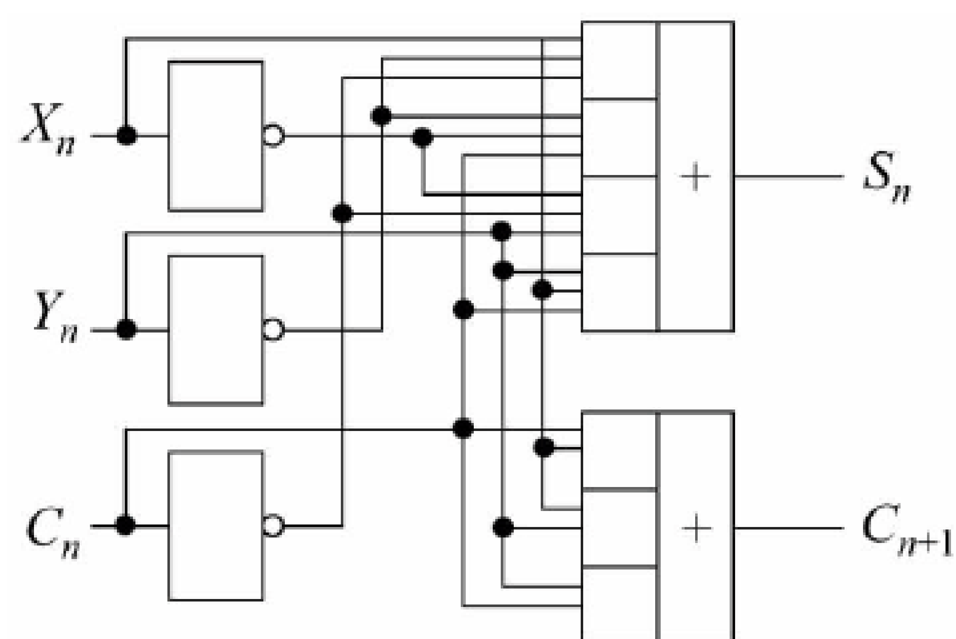
X_n	Y_n	C_n	S_n	C_{n+1}
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

(a) 真值表

$$S_n=\overline{X_n}\,Y_n\,\overline{C_n}+X_n\,\overline{Y_n}\,\overline{C_n}+\overline{X_n}\,\overline{Y_n}\,C_n+X_n\,Y_n\,C_n$$

$$C_{n+1}=X_n\,Y_n\,\overline{C_n}+\overline{X_n}\,Y_n\,C_n+X_n\,\overline{Y_n}\,C_n+X_n\,Y_n\,C_n$$
$$=X_n\,Y_n+X_n\,C_n+Y_n\,C_n$$

(b) 逻辑表达式



(c) 线路图

图 4.1　一位加法器的真值表、逻辑表达式、线路图

如果还希望把 $S_n=X_n$ and Y_n，$S_n=X_n$ or Y_n 的逻辑运算功能也添加进去，需要增加如图 4.2 所示的电路，用一个与门实现与运算，用一个或门实现或运算，再将其与加法电路(抽象为一个矩形框)归并到一起，就得到可完成算术和逻辑运算功能的 ALU。

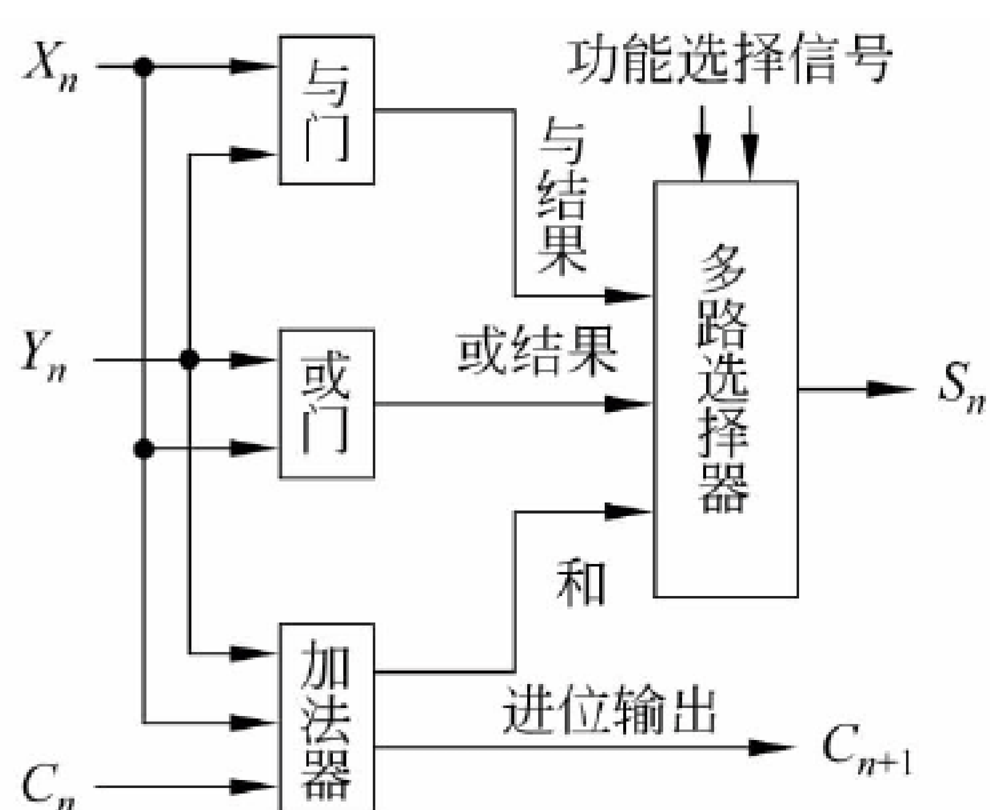


图 4.2　一位 ALU 的原理性组成框图

此时要使用一个多路选择器电路，通过 2 位的功能选择信号(例如信号为 00 输出加法运算的和，为 10 输出“与运算”的结果，为 11 输出“或运算”结果)，从 3 个运算结果中选择其一作为输出。进位信号与逻辑运算无关，只用于加法运算。

在计算机系统中，减法运算是用加法器电路实现的，此时需要把减数每一位取反后送 ALU 的数据输入端，并向 ALU 最低位提供进位输入信号 1。此时需要依据是加运算还是减运算，选择把 Y_n 的值或者 Y_n 每一位取反的值送 ALU，通常用多路选择器电路实现。

请注意，这里给出的只是原理性线路，实际的电路设计要经过精巧的优化处理。

可以用多个一位的 ALU 组成多位的 ALU 部件。图 4.3 给出一个 4 位的 ALU 框图。4 位之间要建立正确的进位连接，低 1 位的进位输出送到相邻高位作为进位输入。

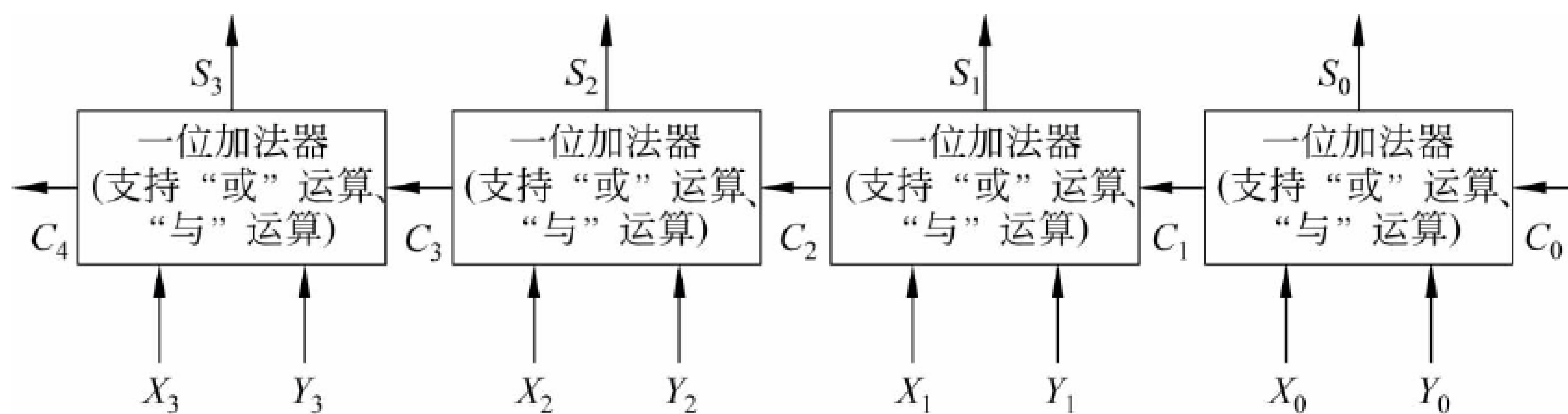


图 4.3　4 位的加法器框图

在多位的 ALU 线路中，如果加法运算的进位信号选用如图 4.3 所示的串行方式传送，用的时间比较长，会影响系统性能。解决的办法是选用超前(并行)进位的方式处理进位信号，实现思路是高位加法器不是简单地等待进位信号从最低位逐位传送过来，而是争取每位加法器**几乎可以同时**得到自己的进位输入信号，看下面的推导过程。

第 n(n 可以取值0～3)位产生进位输出的条件是：2路数据 X_n 和 Y_n 都为1(此时不必关心低位送来的进位信号)，记为 $G_n = X_n \cdot Y_n$，称其为进位产生信号；或者 X_n、Y_n 的和为1，记为 $P_n = X_n + Y_n$，称其为进位传递信号，因为在相邻低位送来的进位信号 C_n 也为1时(表示为 $P_n \cdot C_n$)将产生进位。请注意：这里的符号"·"代表与运算，"+"代表或运算。把位序号0～3代入表达式，可得到：

$$C_1 = X_0 \cdot Y_0 + (X_0 + Y_0) \cdot C_0 = G_0 + P_0 \cdot C_0$$

$$C_2 = X_1 \cdot Y_1 + (X_1 + Y_1) \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$

$$C_3 = X_2 \cdot Y_2 + (X_2 + Y_2) \cdot C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

$$C_4 = X_3 \cdot Y_3 + (X_3 + Y_3) \cdot C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

这表明，4位加法器几乎可以同时得到自己的进位输入信号。这是通过每一位都直接检查本位的运算数据、各低位的运算数据，以及最低位的进位输入信号完成的。其代价是多用了一些处理进位信号的电路。

还可以看到，越往高位，用到的门电路越多，当ALU的位数很多(例如64位)时，就会达到难以承受的程度，可以用分层分组处理并行进位的方式来解决这个问题。例如，把64位ALU划分成4个16位的大组，每个大组再细分成4个4位的小组，参照上述思路，按照3个层次分别处理小组内部、小组之间、大组之间的并行进位逻辑。每个小组、中组、大组都提供本组的 G 和 P 信号，结合 C_0 完成64位的并行进位。这样总体计算下来所用的电路数量并不是很多，*ALU* 的执行速度得到明显提高。

上面的讲解还只限于说明ALU的电路组成和设计原理，到了4.2.2节，我们会给出一个8位字长的原理性的运算器模型，选用ABEL-HDL硬件描述语言进行设计，在现场可编程的CPLD类型的高集成度的芯片内予以实现，达到能够完成运算器某些简单功能教学实验的目标，使学生能真切看到，是如何把一些原理知识和设计技术转换成产品的，又怎样检测产品运行功能的正确性。

4.2 定点运算器

4.2.1 定点运算器部件的功能、组成与控制概述

运算器部件是计算机5大功能部件中的数据加工部件。定点运算器主要完成对整型数据的算术运算和逻辑型数据的逻辑运算功能。运算器的位数取决于机器字长，通常是16位、32位或者64位，将关系到处理数据的能力；完成一次加法运算所用的时间决定了计算机CPU周期的长度，将影响系统的运行速度；运算器内部包含的寄存器个数将影响读写存储器的频率，同样会影响系统的运行速度。由此可见，运算器的组成直接关系到计算机系统的数据处理能力和运行性能，需要认真对待。

1. 定点运算器部件的功能与组成概述

(1) 运算器的首要功能是完成对数据的算术和逻辑运算，由其内部的一个被称为算术与逻辑运算单元(ALU)承担。它在给出运算结果的同时，还给出结果的某些特征，如溢出否，有无进位输出，结果是否为零、为负等。这些结果特征信息通常被保存在几个特定的触

发器中。在执行指令的过程中,必须向 ALU 提供其执行何种运算的控制信号。在一些计算机系统中,这个 ALU 还用于计算指令或者数据在存储器中的地址。

(2) 运算器的第二项功能是暂存将参加运算的数据和中间结果,由其内部的一组寄存器 REGs 承担。因为这些寄存器可以被汇编程序员直接访问,通称通用寄存器,以区别于那些计算机内部设置的、不能被汇编程序人员访问的专用寄存器。为了向 ALU 提供正确的数据来源,必须向寄存器组提供将使用寄存器的编号。

(3) 为了用硬件线路完成乘除指令运算,有些早期的运算器内还有一个能自行左右移位的专用寄存器,通称乘商寄存器。由于该寄存器属于内部专用,汇编程序员不能访问,许多计算机组成原理教材和技术资料中不太提及此线路,后来的计算机不再设置这个寄存器。在 RISC 结构的计算机系统中,乘除法运算选用专门的阵列乘除法部件完成,可以在定点运算中只设置两个保存乘除法计算结果的寄存器。

(4) 这些部件通过几组多路选择器电路实现相互连接,以便数据传送。通用寄存器中的数据移位功能也在运算器内部实现,需要设置相应的电路。

(5) 运算器还要与计算机其他几个功能部件连接在一起并协同运行,就必须有接收外部数据输入和送出运算结果的逻辑电路。

运算器通常还作为处理机内部传送数据的重要通路。

2. 定点运算器的控制与操作概述

如何让运算器完成指定的运算操作功能,是通过向其提供正确的运算数据和控制信号实现的。其包括选择哪一个(几个)数据参加运算,执行何种运算功能,对运算结果(结果的值和特征)如何保存与送出等;同时要解决如何接收外部送来的输入数据,怎样向外部送出运算结果等问题,正确给出 ALU 最低位的进位信号,运算器左右移位操作中的移位输入信号等。用到的控制信号是由计算机的控制器部件提供的,有了这些控制信号,运算器就能完成指定的运算、处理功能。运算器在计算机系统中处于执行部件的地位,受控制器部件的指挥控制。

4.2.2 设计实现一个简单的原理性8位运算器模型

这是一个原理性的最简单的8位运算器模型,还达不到实用水平,但可以把运算器部件的基本组成和功能、把使用 ABEL 语言的基础知识以及使用 CPLD 类型器件的操作过程和相关技术展现清楚。

提供本节教学内容基于两种考虑,对那些未配备我们研制的教学计算机设备的院校,本节内容可供学生阅读,大部分学生应该能够粗浅地读懂其中运算器线路实现的内容,有利于扩展学生视野;对已经或准备配备我们研制的教学计算机设备的院校,本节内容可以用于一次教学实验,学生能够深入具体地学习到更多的知识和实验技术,这些知识与技术在设计与构建教学计算机系统的整个过程中会反复用到。

1. 确定运算器的基本组成和功能

运算器字长8位,ALU 能完成加、减、与、或4种运算,设置一个累计寄存器。

ALU 的两路输入数据分别来自累加器和外部输入数据,计算结果可输出显示,可存入累加器,ALU 要产出标志位信息(进位输出、结果为0),累加器的内容经三态门控制显示。

2. 工程实现的有关选择

要求使用教学机主板上的现场可编程的 ispMACH 芯片把这个运算器实现出来并可以正确运行，运行结果和有关信息可通过指示灯予以显示，输入数据和控制信号用开关拨入。

可以选用硬件描述语言 ABEL-HDL 或 VHDL 来描述 ispMACH 芯片内的电路组成及其功能，我们选择了 ABEL-HDL 语言，这种语言最容易使用，只用到与、或、非三种门电路和 D 型触发器，写出的程序结构简单，清晰易懂。

依据前面的要求，可以画出这个 8 位运算器模型的组成框图，如图 4.4 所示。

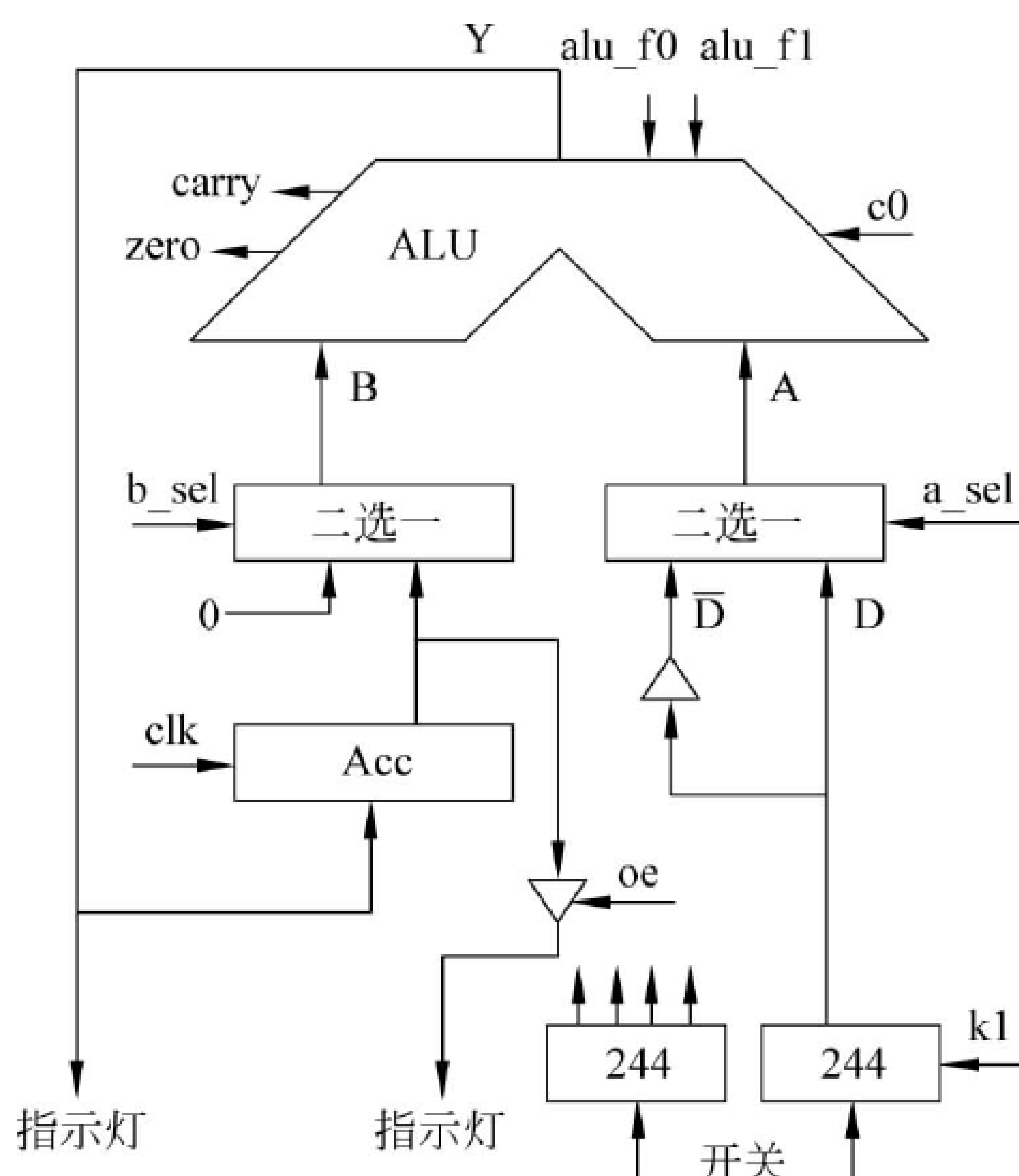


图 4.4　简单运算器组成框图

从图 4.4 中可以看到，该运算器的主要功能电路包括 ALU 和一个寄存器 Acc。

ALU 的 2 路输入是 B 和 A，都是通过二选一电路送来的，B 路数据可以选寄存器 Acc 的输出（实现 Acc 内容累加）或 0（用于向 Acc 赋初值），A 路数据可以选 D（对应非减运算）或 $\overline{D}$（对应减运算），D 是由 8 位开关提供的输入数据；ALU 最低位的进位信号 c0 在运算器内部按需要产生。ALU 的输出包括计算结果 Y（可送到指示灯予以显示）和结果的特征信息（进位输出 carry 和结果为 0 的标志 zero），carry 和 zero 信号也可以送到指示灯予以显示。

寄存器（累加器）Acc 只能接收 ALU 的计算结果 Y，其输出被用作 ALU 的 B 路输入，也能经过三态门控制传送到指示灯予以显示。

该运算器需要用到 4 位的控制信号，alu_f0、alu_f1 用于表示 ALU 的 4 种运算功能（00：+，01：-，10：&，11：#）；b_sel 用于选择 ALU 的 B 路数据来源（0：0 值，1：Acc）；oe 用于选择是否显示 Acc 的内容（0：不显示，1：显示）。

运算器 8 位的输入数据、4 位的控制信号分别来自 8 位开关和 4 位开关，开关的输出端接有三态控制门，只有向这 2 个门电路的管脚 1（信号名为 k）提供低电平的控制信号，开关的输出才能送到运算器的 A 路输入，否则门电路的输出处于高阻状态。

下面给出实现上述运算器设计要求的 ABEL 语言程序,希望大家能够看懂,怎样使用 ABEL 语言来描述电路的组成及其实现的功能,即完成数字电路的设计工作。

3. 描述 8 位运算器的 ABEL 语言程序代码

```
MODULE ispmach "项目模块名
TITLE 'simple alu' "标题名
                              "program alu8_16.abl   2014/10/06   ALU串行进位
DECLARATIONS                 "说明入出信号和内部节点信号
clk                           pin 68;                     "时钟信号
alu_f1,alu_f0,b_sel,oe        pin 64..61;                 "输入 4位控制信号  IR15~IR12
D7..D0                        pin 24,23,26,25,            "输入 8位开关数据  DB15~DB8
                                  28,27,30,29;
Y7..Y0                        pin 32..39;                 "显示 ALU的结果 DB7~DB0
carry,zero                    pin 58, 57;                 "显示 2个特征位 IR9和 IR8
Acc_7..Acc_0                  pin 77..70;                 "显示累加器内容 AB7~AB0
B7..B0,A7..A0,a_sel           node istype 'com';          "ALU的 2路输入数据与选择控制
c8..c1,c0                     node istype 'com';          "ALU每一位的进位入/出信号
Acc7..Acc0                    node istype 'reg,keep';     "8位的累加器
alu_f= [alu_f1,alu_f0];  "定义集合,用于简化逻辑方程式
A= [A7..A0];    B= [B7..B0];    D= [D7..D0];
Y= [Y7..Y0];    Acc= [Acc7..Acc0];
EQUATIONS                      "运算器的功能与电路描述,用了集合
  when b_sel then B= Acc;                                 "ALU的 B路数据选择,条件赋值语句
            else B= [0,0,0,0,0,0,0,0];                    "b_sel= 1选 Acc,否则选 0值
  when a_sel then A= !D;    else A= D;                    "A路数据选择,减运算选 !D,否则选 D
when alu_f== [0,1] then {c0= 1; a_sel= 1;}                "仅在执行减运算时 c0= 1,a_sel= 1
Acc:= Y;      Acc.clk= clk;                               "Acc接收 ALU结果 Y,为 Acc指定时钟
  [Acc_7..Acc_0]= Acc;                                    "累加器内容送指示灯,直接赋值语句
  [Acc_7..Acc_0].oe= oe;                                  "为 Acc指定输出使能信号,oe= 1:可显示
                                                           "ALU的运算功能描述   00:+ , 01:- , 10:&, 11:#
  when (alu_f== [0,0])# (alu_f== [0,1]) then              "加减运算
    { Y0= B0&A0&c0 # B0&!A0&!c0 # !B0&A0&!c0 # !B0&!A0&c0; "逐位计算和/差
      Y1= B1&A1&c1 # B1&!A1&!c1 # !B1&A1&!c1 # !B1&!A1&c1; "结果送指示灯显示
      Y2= B2&A2&c2 # B2&!A2&!c2 # !B2&A2&!c2 # !B2&!A2&c2;
      Y3= B3&A3&c3 # B3&!A3&!c3 # !B3&A3&!c3 # !B3&!A3&c3;
      Y4= B4&A4&c4 # B4&!A4&!c4 # !B4&A4&!c4 # !B4&!A4&c4;
      Y5= B5&A5&c5 # B5&!A5&!c5 # !B5&A5&!c5 # !B5&!A5&c5;
      Y6= B6&A6&c6 # B6&!A6&!c6 # !B6&A6&!c6 # !B6&!A6&c6;
      Y7= B7&A7&c7 # B7&!A7&!c7 # !B7&A7&!c7 # !B7&!A7&c7;
      c1= B0&A0 # B0&c0 # A0&c0;                                "逐位计算进位输出
      c2= B1&A1 # B1&c1 # A1&c1;                                "可以送指示灯显示
      c3= B2&A2 # B2&c2 # A2&c2;
      c4= B3&A3 # B3&c3 # A3&c3;
      c5= B4&A4 # B4&c4 # A4&c4;
      c6= B5&A5 # B5&c5 # A5&c5;
```

```
        c7=B6&A6 # B6&c6 # A6&c6;
        c8=B7&A7 # B7&c7 # A7&c7; }
    when alu_f==[1,0] then   Y=B&A;                    "与运算，用了集合
    when alu_f==[1,1] then
      { Y0=B0# A0; Y1=B1# A1; Y2=B2# A2; Y3=B3# A3;    "或运算，未用集合
        Y4=B4# A4; Y5=B5# A5; Y6=B6# A6; Y7=B7# A7; }  "Y的逻辑方程较多
                         "得到并显示 ALU结果的 2个特征位信息
    when (alu_f==[0,0]) then carry=c8;                 "carry=1:加运算有进位
    when (alu_f==[0,1]) then carry=!c8;                "         减运算有借位
    when [Y7..Y0]==^h00 then zero=1;                   "zero=1:ALU的结果为 0
END
```

4. 对 ABEL 语言程序的简要说明

一个 ABEL 语言程序需要由头段、说明段、逻辑描述段和结束段 4 个基本部分组成。在本例中，程序的前 3 行是头段，从 DECLARATIONS 开始的是说明段，从 EQUATIONS 开始的是逻辑描述段，最后一行的 END 是结束段。

在说明段，需要为 IO 信号分配芯片管脚。程序中每一行的开始是 ABEL 程序中的信号名，再用 pin 指定信号的管脚编号，还可以在注释部分说明信号的功能或用途。注释部分最后几个字是 16 位机系统中使用的信号名，用于找到 MACH 芯片相应管脚在电路板上接线的位置。用 node 说明的是内部节点信号，与 IO 信号的区别是它没有和 MACH 芯片的管脚相连接，通常只在芯片内部使用，在需要输出时，可以通过逻辑方程将其连接到芯片的可用管脚。

信号的类型，组合逻辑信号用 istype 'com'说明，时序逻辑信号(触发器和寄存器)用 istype 'reg,keep'说明，未明确说明的默认为是组合逻辑信号。

在逻辑描述段的开始一些行，分别写出由 b_sel 信号选择向 ALU 的 B 路数据送 0 值(实现向 Acc 赋初值)还是累加器 Acc 的内容(实现 Acc 的累加运算)；在执行减法运算时(alu_f==[0,1])，需要向 ALU 的 A 路数据送入数据 D 的反码，并向 ALU 最低位的进位输入信号 c0 送 1 值，确保减法运算可以使用加法器电路实现。

两个语句 Acc:=Y 和 Acc.clk=clk 完成把 ALU 的计算结果 Y 保存到累加器。

两个语句[Acc_7..Acc_0]=Acc 和[Acc_7..Acc_0].oe=oe 完成在 oe 信号控制下把累加器的内容送指示灯予以显示，这是通过在 Acc 的输出处增加了一个三态控制来实现的。

在逻辑描述段的最后一些行，分别写出实现加、减法运算功能的逻辑方程和实现与、或运算的逻辑方程，以及产生 2 个标志位信息的逻辑方程，这些与教材 4.1 节讲过的内容类似但描述更为精准。

加减法运算的进位信号是逐位传送的，属于串行进位方式，其中的进位输出信号 carry 为 1 时，表明加法计算有进位，减法计算有借位。而逻辑与、或运算在相邻位之间不存在进位关系，故逻辑运算与 carry 信号无关。

请注意，在这个程序中只用到与、或、非门和 D 型触发器，还给出了实现三态控制的办法，在 ABEL 程序中，&、#、! 分别是与、或、非运算符。

初次接触 ABEL 语言程序的人会感到某些困惑，需要对某些概念做出简要说明。

(1) 信号需要在DECLARATIONS段中通过一个标识符予以说明,之后才能在EQUATIONS段中使用。ABEL程序中的信号是对一个硬件对象的抽象,可以是一个逻辑变量或逻辑常量、某种类型的一个数据、一个器件、电路或者一个部件等,到底是什么,需要通过在程序前后文中的说明和使用来确定。

(2) 逻辑型信号是构建数字电路的基石,对逻辑信号最基本的操作是赋值,包括直接赋值和条件赋值两种。直接赋值是把一个逻辑值或一个逻辑表达式的计算结果一定传送给这个信号,条件赋值是在给出的条件成立时才执行本次赋值,还可以指出条件不成立时如何处理。在ABEL语言程序中,接收赋值的信号写在赋值号(=用于组合逻辑电路,:=用于寄存器电路)的左侧,所赋的值(或产生一个值的表达式)写在赋值号右侧。请注意如下两点:

① 计算机中的全部电路都是建立在逻辑运算的基础上,前边给出的8位运算器中的加减法运算是使用逻辑门电路完成的,描述加减法电路使用的是逻辑运算表达式。

② 加减法运算使用的都是补码数,包括开关提供的数据、运算的结果、Acc中的数据都是用补码表示的,这在本教材的第3章已经作为重点内容多次强调过了。

(3) ABEL程序中的语句是同时执行的,与书写的前后次序无关,体现的是同一个电路中的多个器件同时运行的这个事实,而软件算法语言中的语句次序是有实际意义的。

(4) 若有多个语句对同一个信号的赋值操作,其总的效果等同于把每一个赋值语句的执行效果全部按照“或”的关系组合在一起,例如在前面的程序中对每一位Y信号的赋值操作就是如此,把加、减、与、或4种运算(是互斥关系)的结果“或”到了一起。

(5) 需要通过.clk的形式为寄存器电路指定时钟脉冲,D型触发器只在时钟信号的上升沿启动接收操作,例如[Acc7..Acc0].clk=clk;等号左侧的.clk是ABEL语言的扩展属性,等号右侧的clk是用户选用的一个IO信号,此处的两个clk代表的不是同一个事情。

(6) 需要通过.oe的形式为三态门电路提供输出使能信号,例如[Acc7..Acc0].oe=oe;等号左侧的.oe是ABEL语言的扩展属性,等号右侧的oe是用户选用的一个IO信号,此处的两个oe代表的不是同一个事情。

4.2.3 运算器芯片Am2901实例与使用

1. Am2901的内部组成

Am2901芯片是20世纪70年代初的产品,是一个充分反映那个时期集成电路发展水平的4位的位片结构的运算器器件,虽然位数较少但组成完整,作为教学实例具有很好的典型性,其内部组成框图如图4.5所示。使用4片Am2901芯片可构成16位字长的运算器部件。

该芯片的第一个组成成分是一个4位的ALU,它的输出为F,两路输入分别用R和S标记,还有送入ALU最低位的进位信号C_n,能实现$R+S$、$S-R$、$R-S$这3种算术运算功能和$R \vee S$、$R \wedge S$、$\bar{R} \wedge S$、$R \oplus S$、$\overline{R \oplus S}$这5种逻辑运算功能。在给出运算结果的同时,还送出向高位的进位输出信号C_{n+4},溢出标志信号OVR,最高位的状态信号F_3(可用作符号位),以及运算结果为零的标志信号$F=0000$。

该芯片的第二个组成成分是由16个4位的通用寄存器组成的**寄存器组**。它是一个用双端口(A口和B口)控制读出,单端口(B口)控制写入的部件。为了对其进行读写,需通过A地址、B地址指定被读写的寄存器。两路读出数据分别用A口、B口标记,经锁存器线路可以送

到 ALU 的 R、S 输入端的多路选择器，A 口读出数据还可以用作该芯片的可选输出数据之一。寄存器组的写入数据由一组多路选择器给出，并由 B 地址选择写入的寄存器。

该芯片的第三个组成成分是一个 4 位的 **Q 寄存器**，主要用于实现硬件的乘法、除法指令，能对本身的内容完成左、右移位功能，能接收 ALU 的输出，输出送到 ALU 的 S 输入端。

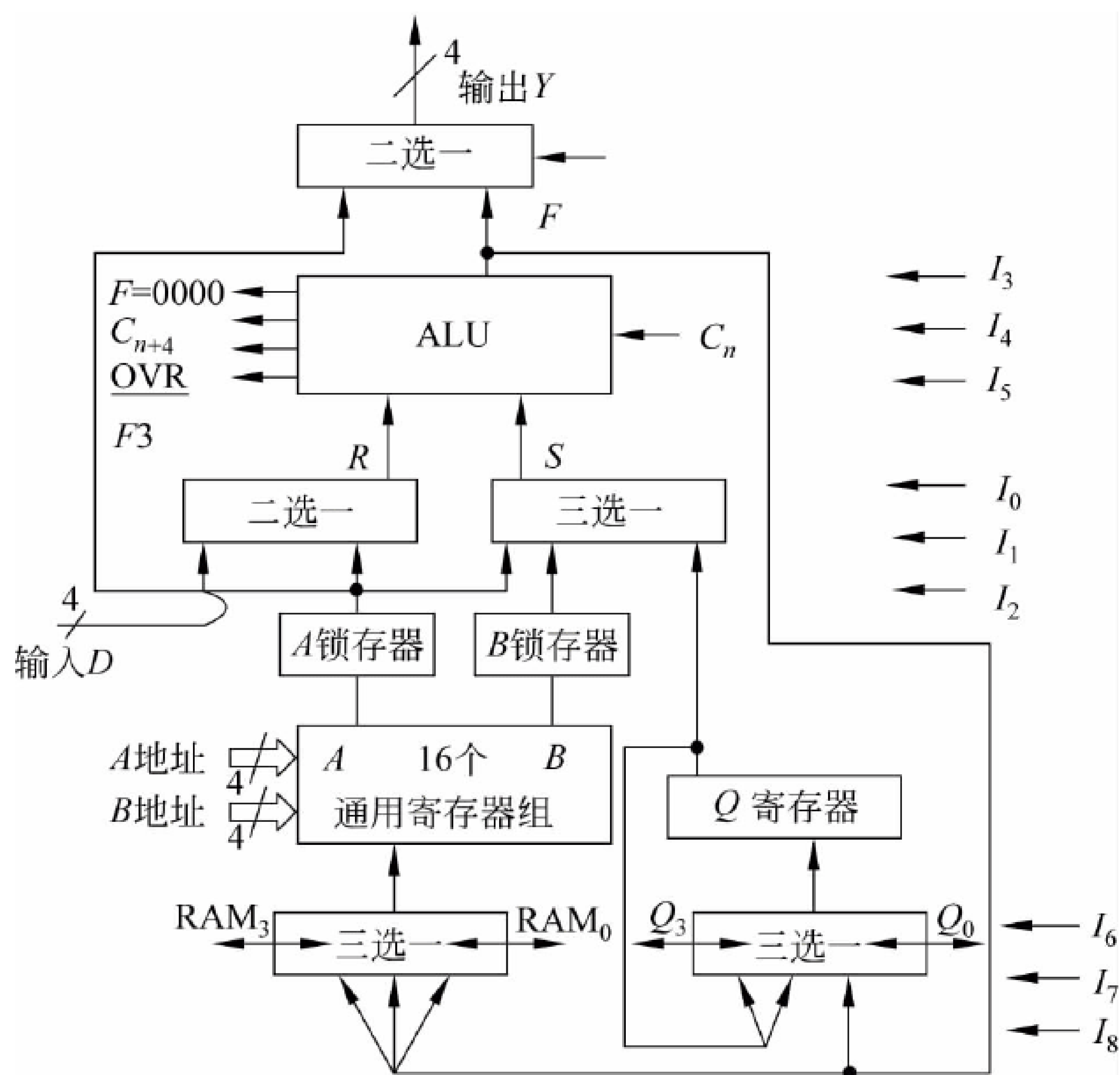


图 4.5　Am2901 芯片的内部组成框图

该芯片的其余组成成分是 **5 组多路数据选择器**，每组都由 4 套电路组成，每套电路对应一个数据位，通过它们实现芯片内部的 3 个组成成分之间的连接，实现芯片内部与外部信息的输入输出操作，包括经 D 输入接收外部送来的 4 位输入数据，经 Y 输出端输出 4 位数据到芯片外部。请注意，在实现寄存器数据左、右移位操作时，会涉及芯片内外部的数据交换，左移操作要求向运算器的最低位(图 4.5 中用 ROM_0 和 Q_0)送入移位输入数据，运算器的最高位(图 4.5 中用 ROM_3 和 Q_3)将向外部送出移位输出数据，右移操作要求向运算器的最高位送入移位输入数据，运算器的最低位将向外部送出移位输出数据，因此运算器的最高位、最低位的移位管脚必须支持三态逻辑，以便用于输入和输出两个方向的数据传送(图 4.5 中用双向的箭头线表示)，通用寄存器移位和 Q 寄存器移位都存在这个问题。

2. Am2901 的控制与操作

为了控制 Am2901 运算器按我们的意图完成预期的运算操作功能，就必须向其提供相应的控制信号和数据。

控制信号包括以下 3 种。

(1) 选择 ALU 的 8 种运算(3 种算术运算、5 种逻辑运算)功能中的一种。这可通过提供 3 位功能选择码 $I_5\ I_4\ I_3$ 实现，其具体规定列在表 4.1 中。

(2) 选择送入 ALU 的两个操作数据 R 和 S 的组合关系(数据来源)。图 4.5 上已标明，R 从 D 和 A 中选择，S 从 A、B 和 Q 中选择，再考虑到它们还均可选 0 值，则我们可以从这许多可能组合中选取最有用的 8 种组合，即 A、Q 组合，A、B 组合，0、Q 组合，0、B 组合，

0、A 组合,D、A 组合,D、Q 组合,D、0 组合,并用 $I_2\ I_1\ I_0$ 这 3 位操作数选择码来控制二组多路选通门选取其一,具体规定如表 4.2 所示。

表 4.1 选择运算功能

编码			运算功能
I_5	I_4	I_3	
0	0	0	$R+S$
0	0	1	$S-R$
0	1	0	$R-S$
0	1	1	$R\vee S$
1	0	0	$R\wedge S$
1	0	1	$\overline{R}\wedge S$
1	1	0	$R\oplus S$
1	1	1	$\overline{R\oplus S}$

表 4.2 数据来源选择

编码			数据来源	
I_2	I_1	I_0	R	S
0	0	0	A	Q
0	0	1	A	B
0	1	0	0	Q
0	1	1	0	B
1	0	0	0	A
1	0	1	D	A
1	1	1	D	Q
1	1	1	D	0

表 4.1 中的符号:“∨”表示或运算,“∧”表示与运算,“⊕”表示异或运算,“—”表示取反运算。

(3) 选择运算结果或有关数据以什么方式送往何处的处理方案,这主要指通用寄存器组和 Q 寄存器执不执行接收操作或移位操作,以及向芯片的输出信息 Y 提供的是什么内容。这是通过 $I_8\ I_7\ I_6$ 这 3 位结果选择码来控制 3 组相应的选择门实现的,其规定如表 4.3 所示。3 位控制信号为 000,通用寄存器内容不变,Q 寄存器接收输入;为 001,通用寄存器和 Q 都不接收输入;为 010 或 011,通用寄存器接收输入,区别在于向芯片外送出的分别是 A 口读出内容或 ALU 的计算机结果;为 101 或 111,都是通用寄存器接收输入,区别在于接收的信息分别是 ALU 的计算结果向右或者向左移一位的内容,用在实现对寄存器内容执行移位的指令之中。现在已经不再使用 Q 寄存器,故不会用到 000、100、110 这 3 个编码。

表 4.3 选择结果处理方案

编码			结果处理		
I_8	I_7	I_6	通用寄存器组	Q 寄存器	Y 输出
0	0	0		$F\to Q$	F
0	0	1			F
0	1	0	$F\to B$		A
0	1	1	$F\to B$		F
1	0	0	$F/2\to B$	$Q/2\to Q$	F
1	0	1	$F/2\to B$		F
1	1	0	$2F\to B$	$2Q\to Q$	F
1	1	1	$2F\to B$		F

关于该芯片的具体线路尚需以下几点说明。

(1) 芯片输出数据还受一个 OE 信号的控制，仅当其为低电平时，才有 Y 值输出，否则输出为高阻态。标志位 F=0000 为集电极开路输出，可实现“线与”逻辑，此管脚需经一电阻接到+5V。RAM_3、RAM_0、Q_3、Q_0 均为双向入出管脚，一定要与外部电路正确连接。通用寄存器组通过 A 端口、B 端口读出内容均经锁存器缓存，保证执行诸如 $A+B$ 结果送入 B 运算操作的正确性。

(2) 该芯片还有两个用于芯片之间完成超前进位的输出信号 $\bar{G}$ 和 $\bar{P}$，我们未使用。

(3) Am2901 芯片要用一个 CLK(CP)时钟信号作为芯片内通用寄存器、锁存器和 Q 寄存器的时钟信号。其有关规定如图 4.6 所示。注意，CLK 两个跳变沿和低电平所实现的不同控制功能。

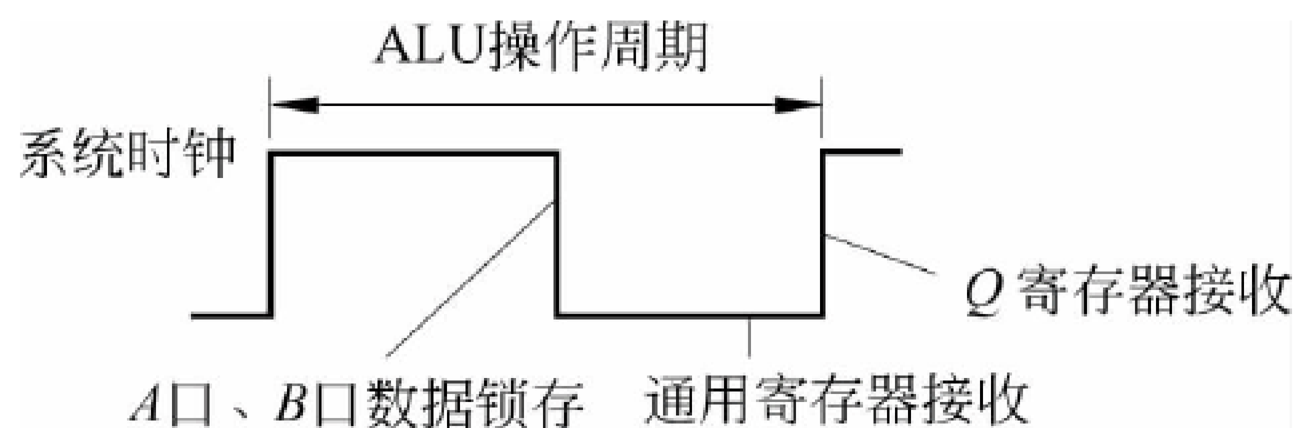


图 4.6　Am2901 的时钟信号 CLK(CP)的作用

3. 外部的数据及线路

有一些数据是由在芯片之外的线路提供的，包括：

① 芯片经 D 端接收的外部数据；

② 芯片最低位的进位输入信号 C_n；

③ 关于左右移位操作过程中的 RAM_3、RAM_0、Q_3 和 Q_0 的输入数据；

④ 需要在芯片外设置接收与记忆 4 个标志位(Flag)信息的电路。

4. 用 4 片 Am2901 芯片构建 16 位的运算器部件

教学计算机的字长是 16 位，为此需要选用 4 片 Am2901 芯片构建运算器部件，其连接关系如图 4.7 所示。

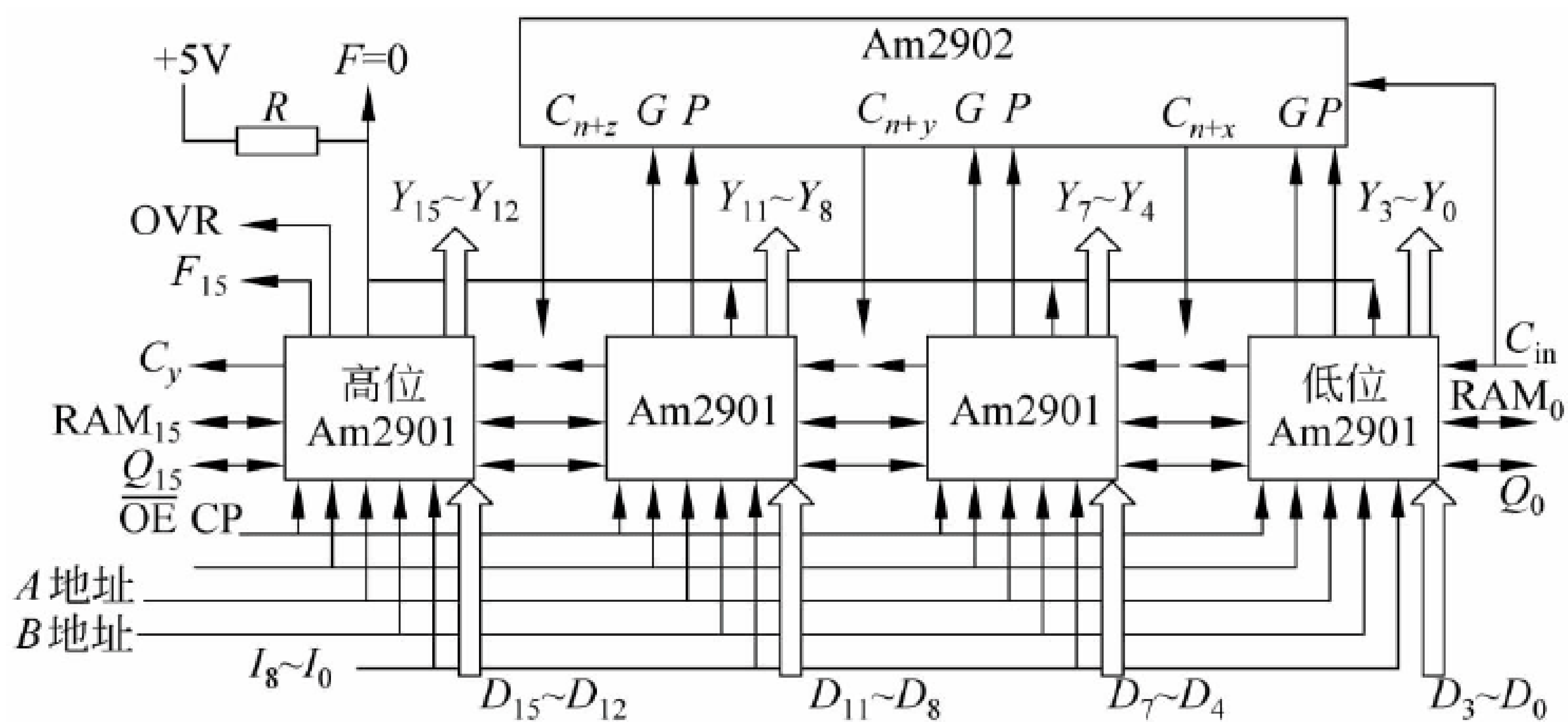


图 4.7　4 片 Am2901 芯片的级联

图中表明 4 个芯片之间可以选择串行进位或并行进位(要用到 Am2902 芯片)。从这里可以看到前面讲到的并行进位、进位产生信号 G、进位传递信号 P 及其应用实例。

最低位芯片的 C_n 是整个运算器的最低位进位输入最高位芯片的 C_y 是 16 位运算器的

进位输出。只有最高位芯片的 F_3 和 OVR 有意义,低位的3个芯片的 F_3 和 OVR 不被运用。

4个芯片的 F=0000 管脚(集电极开路输出)连接在一起,并经一个电阻接到+5V 电源,以得到16位 ALU 的运算结果为0的标志位信号。

其他的几组输入信号,对4片 Am2901 器件来说应有相同的值,包括 OE(控制 Y 的输出)和工作脉冲 CP,A 地址、B 地址(寄存器编号),$I_8 \sim I_0$(控制 Am2901 的结果处置,运算功能,数据来源),故应将这4个芯片的这些对应信号的管脚连接在一起,如图4.7所示,请注意图中的某几个引脚的名字有所变化。

这个运算器芯片实例将用于构建教学实验设备的运算器部件,因此在教师授课和学生学习的过程中应给予更多的关注。要学懂这个运算器部件的内部组成和使用方法,这也是学好控制器的必要准备,请不要掉以轻心。

对于 Am2901 运算器芯片的特点总结如下。

(1) 这是一个组成比较完整的运算器,内含 ALU、16个寄存器组成的通用寄存器组、用硬件完成一位乘除法运算的乘商寄存器 Q、支持寄存器左右移位的电路,用于教学有很好的典型性,虽然单个芯片只有4位字长,但可以很方便地用4个芯片组成16位的运算器部件。其接收外部输入数据和输出运算结果、送出状态位信息的办法简单。

(2) 这个运算器使用的控制信号组织合理,用3组3位的控制信号分别控制 ALU 以下3个方面的操作功能:① ALU 的运算功能。② ALU 两路输入数据的来源和组合关系。③对本步骤执行结果的处理方案,包括寄存器是否接收 ALU 的运算结果、是否伴有移位动作。向外送出的16位数据可以是 ALU 的运算结果,也可以是通过 A 口读出的一路运算数据,这在修改堆栈指针 sp(即 R_4)时很有用,可以用一个时钟周期完成 sp−1→ar 和 sp−1→sp 的操作(用于堆栈写入),或者 sp→ar 和 sp+1→sp 的操作(用于堆栈读出)。在芯片内部,对向芯片外送出的16位数据都加了用 OE 信号控制的三态门,可简单地解决与其他电路分时使用 DB 的冲突,当 OE=0 时允许送出数据到 DB,OE=1 时这16位数据的输出管脚呈高阻态,不影响其他电路使用 DB。

(3) 选用寄存器组中的哪一或两个寄存器,是通过向寄存器组提供一或两个4位的寄存器编号(称为 A 口地址、B 口地址)实现的,最为简单便捷。请注意,执行读寄存器组操作时,可以通过 A 口地址、B 口地址同时读出两个寄存器的内容,但在执行写寄存器组操作时,只能把 B 口地址用作接收写入的寄存器的编号。

(4) 需要向芯片提供时钟信号,用于解决芯片内部电路的时序关系,无须系统的设计者执行其他辅助处理。这就意味着,只要向运算器提供了正确的17位的控制信号,运算器就能够完成使用者所要求的运行功能,不必担心遇到线路的竞争冒险的问题。

(5) 这个运算器中能够在一个时钟周期完成一次数据运算功能,可以有效地减少指令的执行步骤,降低了控制器部件的设计难度,比起下一节要讲到的 MIPS32 计算机系统中的运算器简单了许多,相应地降低了学生学习的难度。这也是我们选择 Am2901 芯片构建教学计算机系中的运算器部件的理由之一。

4.2.4 MIPS 多指令周期 CPU 系统的运算器的组成及其功能

1. MIPS32 系统中的基本运算器部件

MIPS 计算机是20世纪80年代中期推出的典型 RISC 结构,是一个非常成功的系统,

国内外许多教材都把 MIPS 的指令系统和实现技术选为教学内容。MIPS 计算机的设计目标之一是实现通畅的指令流水，这一目标在运算器部件中得到充分体现。我们将以 32 位字长的机型为例，介绍 MIPS 计算机的运算器部件。其内部基本组成如图 4.8 所示，包括两个重要部分，一个是由 128 个寄存器组成的寄存器组，另一个是执行数据运算的 ALU。这个运算器被用于多周期 CPU 系统（对不同类型的指令选用不同数目的执行步骤）时，ALU 既用于计算数据，又用于计算数据和指令在存储器中的地址，故还需要向 ALU 提供计算指令地址的相关信息。

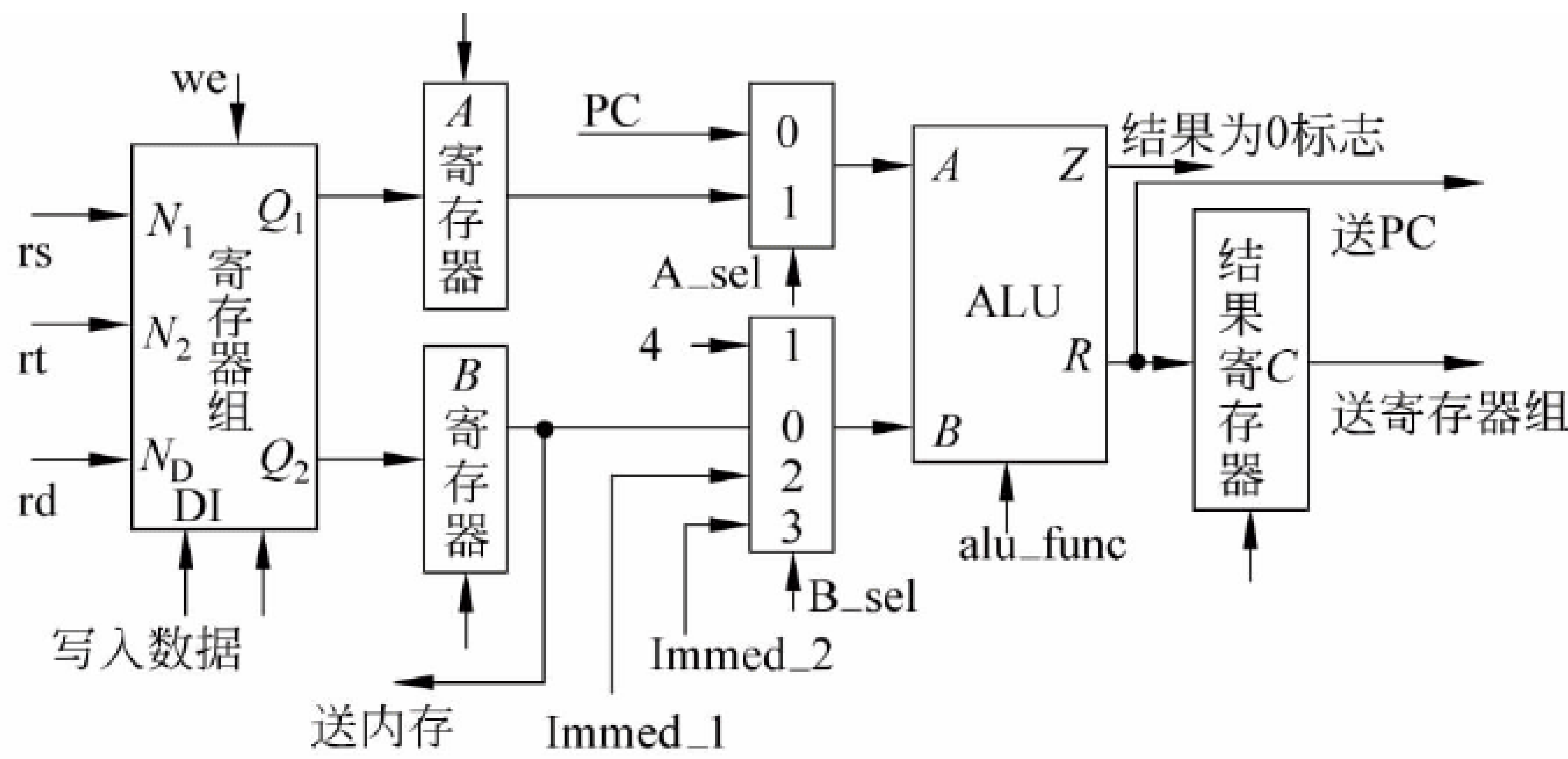


图 4.8　MIPS 计算机的运算器部件

寄存器组可以使用两个端口（N_1、N_2）控制读出，从寄存器组中读出两个寄存器的内容（Q_1、Q_2）将送到寄存器 *A* 和 *B*，作为 ALU 运算的两路数据来源，*B* 寄存器的内容还可以作为写入内存的数据。寄存器组还要用第三个端口（N_D）提供写入数据时用到的寄存器编号。

ALU 完成对两路输入数据（*A*、*B*）的算术或者逻辑运算功能，可向外直接提供运算的结果 *R*（送 PC）和结果的特征信息 *Z*（结果为 0 标志），或将结果暂存于结果寄存器 *C* 中，为后续操作准备数据。需要向 ALU 提供不同来源的两路数据，*A* 路可以选择程序计数器 PC 或者寄存器 *A* 的输出，*B* 路可以选择常数 4、寄存器 *B* 的输出或者另外两路数据 Immed_1 和 Immed_2。这可以组合出 *A* OP *B*，*A* OP Immed_1（OP 代表不同的算数或者逻辑运算），PC＋4，PC＋Immed_2 等多种运算功能。前两种组合用于数据运算，包括两个寄存器之间的运算，或者一个寄存器和立即数之间的运算；后两种组合用于计算指令地址，PC＋4 得到下条相邻指令的地址，PC＋Immed_2 得到相对转移指令的地址，更深入的内容将在控制器部件的章节进行说明。

2. 运算器部件的工作过程和需要的控制信号

请注意，在这个运算器部件中完成一次数据运算要用 3 个时钟周期，有利于实现指令流水。首先是从寄存器组读出寄存器内容并缓存（需要提供被读的两个寄存器的编号 rs 和 rt），之后 ALU 执行运算并暂存（需要提供数据来源选择信号 A_sel 和 B_sel，以及 ALU 的运算功能选择信号 alu_func），最后一步才会把暂存在寄存器 *C* 中的计算结果写回寄存器组（需要提供被写的寄存器编号 rd，写寄存器组的命令信号 we）。寄存器组和 3 个寄存器 *A*、*B*、*C* 都要用到时钟信号 Clock。这与 Am2901 运算器用一个时钟周期完成一次数据运算不同。

有关这个运算器的其他内容，将到 6.2.2 节的 MIPS 的控制器部件实例中进一步说明。

4.3 浮点运算和浮点运算器

浮点运算器是主要用于对计算机内的浮点数进行运算的部件。浮点数通常由阶码和尾数两部分组成,阶码是整数形式的,尾数是定点小数形式的。这两部分执行的操作不尽相同。因此,浮点运算器总是由处理阶码和处理尾数的这样两部分逻辑线路组成。本节将从浮点数运算的实现算法和浮点运算器的组成特点两个方面,来讨论浮点运算器的有关内容。为了平衡各章节的教学负担,我们没有把浮点数的表示和运算的实现算法放在第3章,而是拿到这里和浮点运算器安排在一起进行讲解。

4.3.1 浮点数的运算规则

前面已讲到,浮点数通常被写成如下形式:

$$X = M_x \times 2^{Ex}$$

浮点数通常有两种表示方式,一种表示方式用于运算过程,出现在浮点运算器内部,此时的 M_x 是浮点数的尾数,机器中多用原码表示,不使用隐藏位技术,是绝对值小于1的规格化的二进制小数。E_x 为该浮点数的阶码,机器中多用移码表示,一般为二进制整数,给出的是浮点数表示中的实际指数的幂+128(8位阶码时),而该指数的底通常选用2。另外一种表示用于浮点数的存储过程,对于规格化的非0值的浮点数使用隐藏位技术,即隐去规格化尾数最高一位上的数值1,剩余尾数再左移一位,并相应地把阶码部分的值减1,变为实际指数的幂+127(8位阶码时)。这样做既不影响实际运算,保存过程中尾数又多一个二进制位的精度。

1. 浮点加减法的运算步骤

假定有两个浮点数

$$X = M_x \times 2^{E_x}, \quad Y = M_y \times 2^{E_y}$$

实现 $X \pm Y$ 运算,要用如下多步完成。

(1) 计算阶差操作,即比较两个浮点数的阶码值的大小,求 $\Delta E = E_x - E_y$。

(2) 当阶差 ΔE 不等于零时,首先应使两个数取相同的阶码值。其实现方法是,把原来阶码小的数的尾数右移 $|\Delta E|$ 位,其阶码值取大的阶码值,则该浮点数的值不变(但精度变差了)。为减少误差,可用另外的线路,保留右移过程中移出去几位值,用于以后舍入。

(3) 实现尾数的加(减)运算,即对两个完成对阶后的浮点数执行求和(差)操作。

(4) 规格化处理,完成对不满足规格化规则的计算结果的规格化操作。

(5) 有时还需要执行舍入操作。舍入的总的原则是要有舍有入,而且尽量使舍和入的机会均等,以防止误差积累。常用的办法有0舍1入法,即移掉的最高位为1时,则在尾数末位加1;移掉的最高位为0时,则舍去移掉的数值。该方案的最大误差为 $2^{-(n+1)}$。另一种方法是恒"置1"法,即右移时,丢掉移出的那些低位上的值,并把结果的最低位置成1。还有更精确的舍入处理方案,在此不予介绍。

(6) 接下来还要判断结果的正确性,即检查溢出。浮点数的溢出是以其阶码溢出表现出来的。在加减运算真正结束前,要检查是否产生了溢出,若阶码正常,加(减)运算正常结束;若阶码下溢,要置运算结果为浮点形式的机器零,若上溢,则置溢出标志。

规格化浮点加减运算流程如图 4.9 所示。

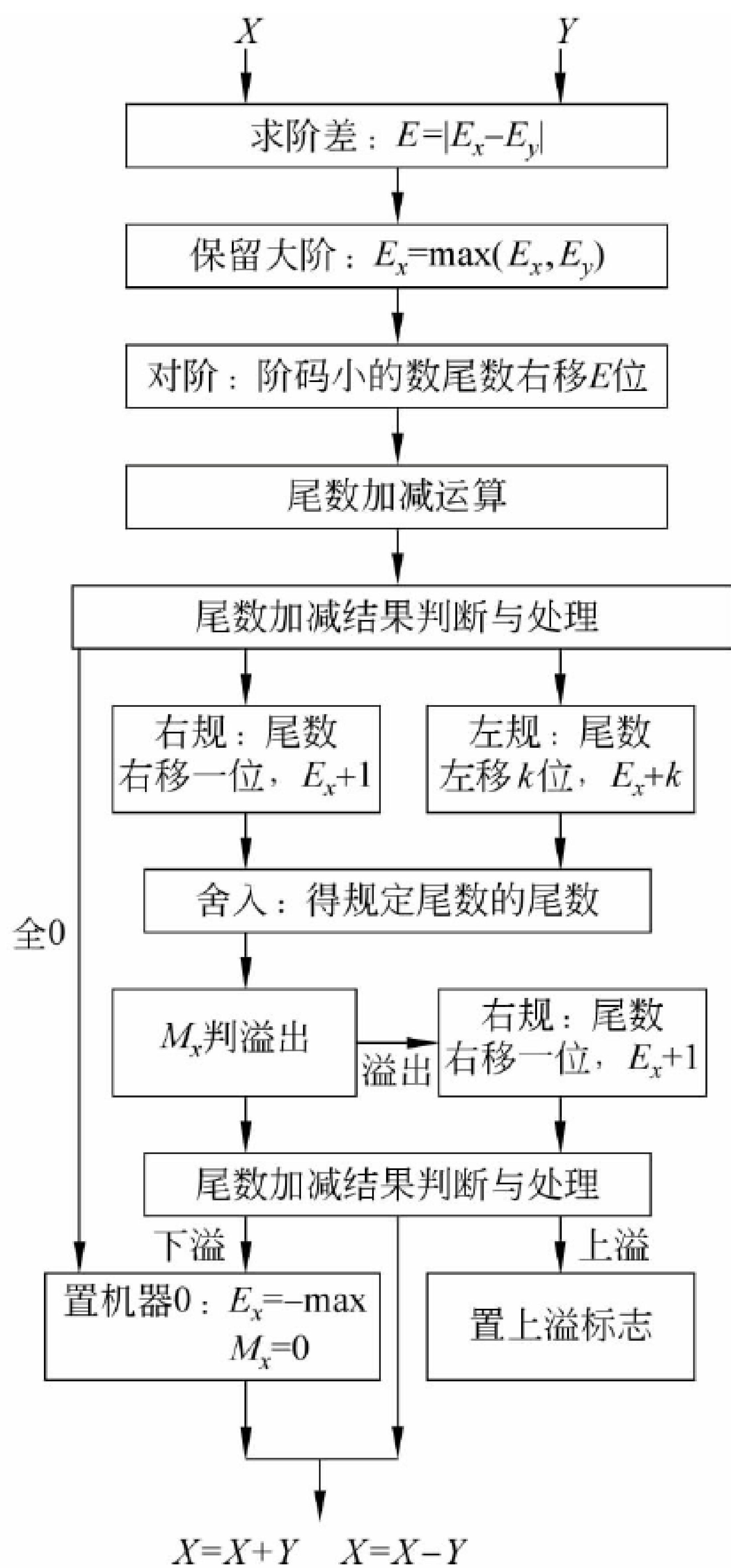


图 4.9 规格化浮点加减运算

看一个浮点数加法运算的实例。假定浮点数的阶码为 4 位，选用 +8 的移码表示，尾数为 8 位(含符号位)，用原码表示。则当 $X=2^{010}\times 0.1101111$，$Y=2^{100}\times(-0.1010110)$时，则它们的浮点数表示分别为：

$[X]_{补}=0\ 1010\quad 1101111$ （1 位符号位，4 位阶码，7 位尾数数值位）

$[Y]_{补}=1\ 1100\quad 1010110$ （1 位符号位，4 位阶码，7 位尾数数值位）

执行 $X+Y$ 的过程如下。

(1) 求阶差对阶。$E=E_x-E_y=[E_x]_{移}+[-E_y]_{移}=1\ 010+1\ 100=1\ 110$，即 ΔE 为−2。

请注意，移码数相加是对相加得到的符号位求反后才是正确移码符号位的值。

(2) 对阶。X 的阶码小，应使 M_x 右移两位，E_x 取值为 4，得$[X]_{浮}=1100\ 0\ 001101111$，用另外的线路保存 X 尾数的最低 2 位，用于舍入。注意，本步骤已把数的符号移到尾数数值位之前了。

(3) 尾数求和。此处是原码相加，执行的是 M_y-M_x。

$$
\begin{array}{r}
0\ 0011011\ \boxed{11} \\
+1\ 1010110\ \phantom{\boxed{11}} \\
\hline
1\ 0111010\ \boxed{01}
\end{array}
$$

(4) 规格化处理。尾数的最高位为0,应执行左规1位处理,结果为1 111010010,阶码减1,为1 011。

(5) 舍入处理。采用0舍1入法处理,把最低位之后警戒位上的1加到尾数中,最终结果为1 1110101。

(6) 判溢出。阶码为1 011,不溢出。将此结果还原成正确的浮点数表示,则得1 1011 1110101,故得最终结果为 $X+Y=2^{+011}\times(-0.1110101)$。

2. 浮点数乘、除法运算步骤

浮点数相乘,乘积的阶码应为相乘的两数的阶码之和,尾数应为相乘两数的尾数之积。

浮点数相除,商的阶码应为被除数的阶码减去除数的阶码得到的差,尾数应为被除数的尾数除以除数的尾数所得的商。规格化浮点乘除运算流程如图4.10所示。

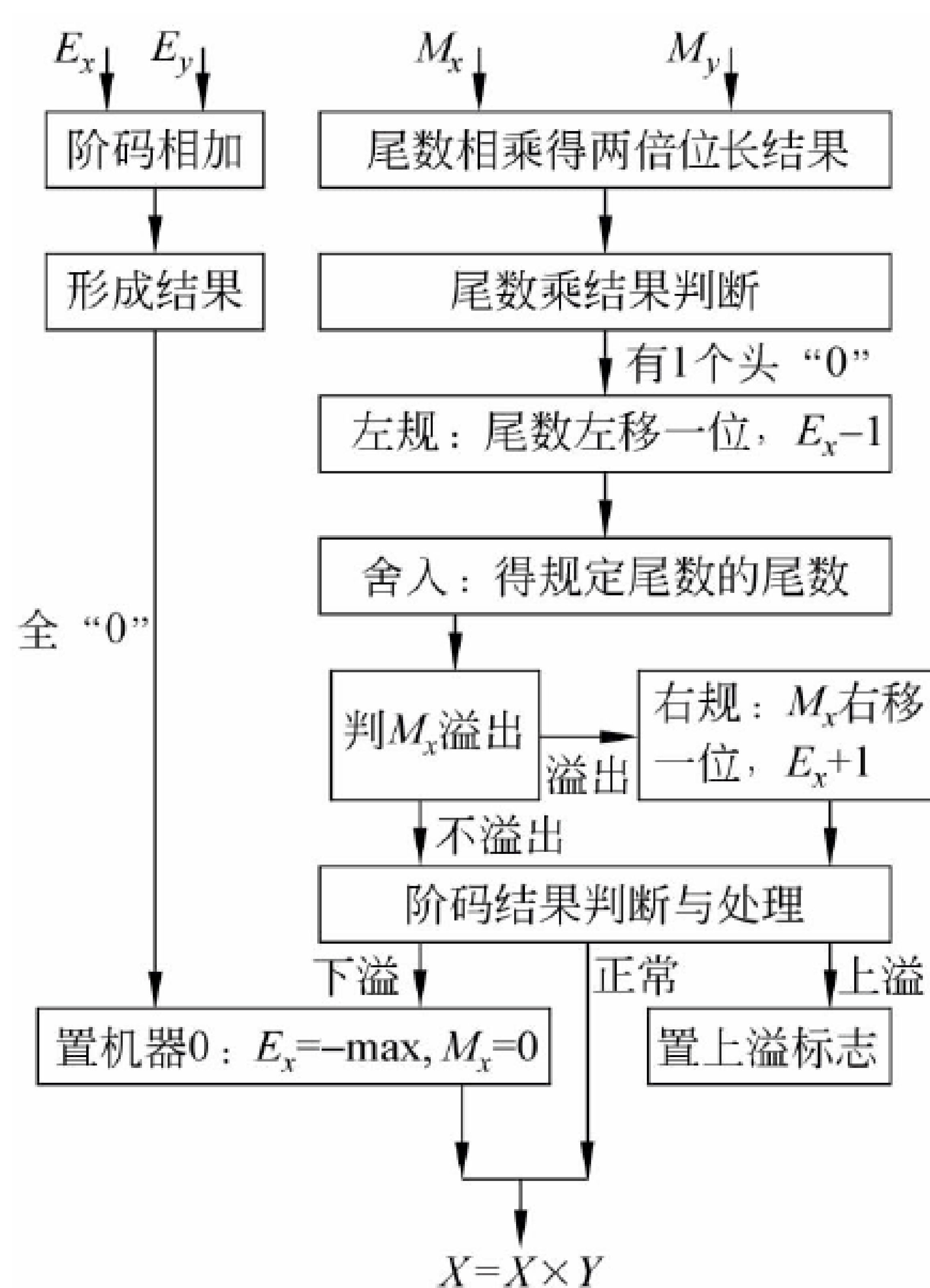


图4.10 规格化浮点乘法运算

乘、除运算都可能出现结果溢出或结果不满足规格化要求的问题,因此也必须进行这些检查和处理。与加减法运算类似的是,乘、除法的浮点数运算也有个精度处理要求,舍入矛盾似乎更突出一点。下面就阶码运算和尾数舍入问题进行讨论。

1) 浮点数的阶码运算

对阶码的运算主要有+1、−1、阶码求和、阶码求差4种,运算时还必须进行溢出检查。在计算机中,阶码通常用移码形式表示,当对 n 位的移码选用移 $2^{(n-1)}$ 的方案时,移码的定义为

$$[X]_{移}=2^n+X\quad -2^n\leqslant X<2^n(\mathrm{Mod}\quad 2^{n+1})$$

按此定义,则有

$$
\begin{aligned}
[X]_{移}+[Y]_{移}&=2^n+X+2^n+Y=2^n+(2^n+X+Y)\\
&=2^n+[X+Y]_{移}
\end{aligned}
$$

上述表达式表明,在直接用移码实现求和运算时,结果的最高位(符号位)多加了个1,要得到正确的移码结果,必须对计算得到的符号再执行一次求反操作。

2) 浮点数的尾数处理

在计算机中,浮点数的尾数总是用确定的位数来表示的,但浮点数的运算结果却常常超过给出的位数,如加减运算过程中的对阶和右规处理,会使尾数低位部分的一位或多位的值丢失;乘除运算也可有更多位数的结果,有多种办法处理多出来的这些位上的值,称为舍入。

第1种办法,是无条件地丢掉正常尾数最低位之后的全部数值,称为截断处理,处理简单但影响结果的精度。

第2种办法,运算过程中保留右移中移出的若干高位的值,最后再依据这些位上的值修

正尾数,称为舍入处理,有利于提高结果的精度。

对原码表示的正、负数的尾数,舍入规则比较简单。最简便的是0舍1入法;也可以只要尾数最低位为1,或移出的几位中有为1的数值位,就使尾数最低位的值为1。

下面给出浮点数相乘的一个实际的例子。假定浮点数的阶码为4位的移码,尾数(含符号位)为8位的原码,并规定不用隐含位,阶码的底为2,乘法的结果尾数仍保留8位(含符号位),直接用原码完成尾数的乘法运算,用0舍1入方法进行舍入。

假定 $X=2^{-5}\times 0.1110011$,$Y=2^{3}\times(-0.1110010)$,则二数的浮点表示为

$$[X]_{浮}=0\ 011\quad \mathbf{0}\ 1110011,[Y]_{浮}=1\ 011\ \mathbf{1}\ 1110010$$

$X\times Y$ 的执行步骤如下。

(1) 乘积的阶码为二数阶码之和,用公式 $[E_x+E_y]_{移}=[E_x]_{移}+[E_y]_{移}$,则有 0 011+1 011=0 110,结果为移码表示的−2。

(2) 原码尾数相乘的结果为 **1** 1100110**0110**110。

(3) 规格化处理,已满足规格化要求,不需左规,尾数不变,阶码仍为0110。

(4) 舍入处理,按舍入规则,尾数之后的4位为0110,可以直接舍掉。

所以相乘结果的最终值为 $[X\times Y]_{浮}=1\ 0110\quad 1100110$,其真值为 $2^{-2}\times(-0.1100110)$。

执行浮点数除法的计算与浮点数乘法的运算步骤类同,此处不再赘述。

3. 有关浮点数运算和浮点运算器的一点补充说明

1) 阶码的底为8或16的情况

前面的讨论都是以阶码值的底为2来进行的。为了用相同位数的阶码表示更大范围的浮点数,在一些计算机中,也有选用阶码的底为8或16的情况,此时被表示成

$$X=8^{E_x}\times M_x \quad 或 \quad X=16^{E_x}\times M_x$$

此时阶码 E_x 和尾数 M_x 还都是用二进制表示的,其运算规则,与阶码以2为底时基本相同,但关于对阶和规格化操作必须有新的相应规定。

当阶码以8为底时,只要原码尾数满足 $1/8\leqslant|M_x|$ 就是规格化表示形式,即尾数数值的最高3位中至少要含有1位1。执行对阶和规格化操作时,每当阶码的值增1或减1,尾数要相应地移3位。

当阶码以16为底时,只要原码尾数满足 $1/16\leqslant|M_x|$ 就是规格化表示形式,即尾数数值的最高4位中至少要含有1位1。执行对阶和规格化操作时,每当阶码的值增1或减1,尾数要相应地移4位。

2) 浮点数的阶码用移码和尾数用原码表示的优点

尾数的符号位在浮点数表示的最高位,比较两个浮点数的大小时,符号非常重要,正数一定大于负数。阶码的位置在机器数表示中,处在符号位和尾数值之间,阶码值大的,其移码形式的机器数也大,与原码表示的尾数配合起来更便于比较浮点数的大小。

移码的最小值是各位均为0。它通常可以被用来表示机器零,即当阶码的值小于或等于移码所能表示的最小值时,认为浮点数的值为0。此时的机器0为阶码和尾数均为0的形式,给硬件的判零带来很大方便。

4. 关于 IEEE 浮点数标准 754 的有关规定

从简化课程讲授和易于理解的角度,在前面对浮点数的文字说明和运算实例中,都是围绕基本原理进行的。当浮点数的阶码为8位时,采用的是移128的方案,也没讨论在尾数中

使用的隐藏位技术,是比较传统的做法,这与IEEE浮点数标准754的实际规定有一些差别。在IEEE浮点数754标准中,规定浮点数的尾数使用原码表示,对规格化的非0值尾数使用隐藏位技术。对短(单精度)浮点数(阶码8位)采用的是移127的移码方案,对长浮点数(阶码11位)采用的是移1023的移码方案。

原码尾数中使用隐藏位技术是指把非零值的规格化浮点数的尾数最高位上的1强行去掉(隐藏起来)。这是通过左移原来的尾数实现的。其效果是使结果的表示精度多出了1个二进制位,短浮点数的有效精度从23个二进制位变到24个二进制位。考虑到隐藏位和剩余尾数,则此时它所代表的实际值在1～2之间。

在移127的移码方案中,8位移码结果不再与8位的补码存在仅符号位相反的对应关系。其值要通过对阶码的实际值加127来得到,即当阶码分别为－126、－125、－2、－1、0、＋1、＋2和＋127时,8位的移码是00000001、00000010、01111101、01111110、01111111、10000000、10000001和11111110这8个数字。可以表示的规格化浮点数的阶码值在－126～＋127之间。

移码形式的阶码值00000000和11111111被分派特定的用处。当移码值为00000000时,过去简单的处理办法是强行置尾数结果也为0值,就是过去说的机器0;到了IEEE标准,则可以用其表示非规格化浮点数。此时不能再使用隐藏位,只要尾数23位上不是全0,都属于能表示的、非规格化的浮点数,这比过去认为这些数就是0值是一个改进。

移码形式的阶码值为11111111时,同样不能用于表示规格化浮点数,把它与全0值的尾数结合起来,用于表示无穷大的值。当尾数的符号为正时,表示正无穷大的浮点数,当尾数的符号为负时,表示为负无穷大的浮点数。

当阶码和尾数的所有位都是0时,代表浮点数的0值,此时的浮点数有＋0和－0两种形式,依据浮点数的符号位是0还是1来加以区别。

计算一个IEEE标准的规格化浮点数的实际值的公式为

$$(-1)^S \times (1+\text{剩余尾数}) \times 2^{(\text{阶码值}-127)}$$

此处的S是浮点数表示中的符号位,0正1负;$(-1)^0$为1,$(-1)^1$为－1。

(1＋剩余尾数)中的1是隐藏位上的值,代表正1;剩余尾数(去掉隐藏位后的尾数)是纯小数部分,二者之和大于等于1并且比2略小。

(移码形式的阶码值－127)是实际的阶码值,可正可负,在－126～＋127之间。

这与前面原理上介绍过的计算一个浮点数实际值的公式有如下对应关系:

$$N=M\times 2^E$$

通过不使用隐藏位、阶码移128的浮点数表示计算浮点数实际值的公式:

$$N=(-1)^S\times M\times 2^{(E-128)}$$

可以发现,在不使用隐藏位时,全部尾数都在小数点之后,比使用隐藏位的值小了将近一半,故此处的阶码值应该比使用了隐藏位情况下的阶码值大1,所以两种公式的计算结果是相同的,但使用了隐藏位的计算精度可以多出1个二进制位。

4.3.2 浮点运算器举例

本节将以Intel 80287浮点协处理器为例,来讨论浮点处理器的组成和运行方式。80287是Intel公司生产的、为配合Intel 80286或80386微处理器处理浮点数等而设计的选

购件。在当前流行的 Pentium 微处理机系统中，性能更高的浮点运算器已集成到 CPU 芯片内。

1. 80287 的性能及内部结构

80287 内部结构的逻辑框图如图 4.11 所示。

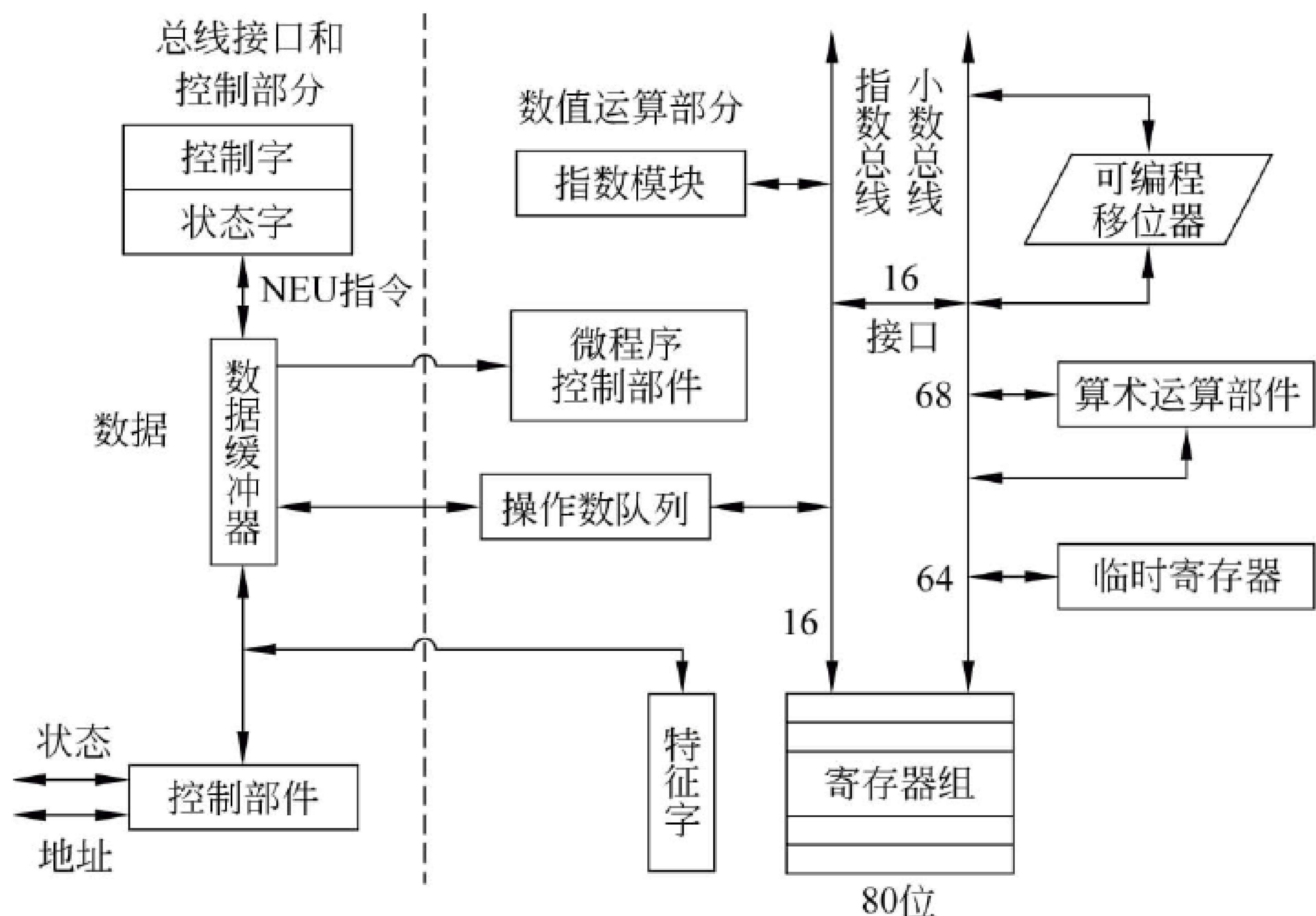


图 4.11　80287 内部结构的逻辑框图

80287 浮点协处理器的主要性能如下。

(1) 可与 80286 或 80386 异步并行工作。

(2) 高性能的 80 位字长的内部结构，有 8 个 80 位字长的以堆栈方式管理的寄存器组。

(3) 浮点数的格式，完全符合 IEEE 制定的国际标准。

(4) 能处理包括二进制浮点数、二进制整数和十进制数串 3 大类共 7 种数据。

(5) 扩展了 80286 或 80386 的硬件指令，直接支持对 7 种数据的指数、对数、三角函数和其他一些数学函数的计算。

(6) 内部的出错管理功能。

80287 是被做在单个芯片之内，并用 40 条引线的外壳封装。它不是一个简单的浮点运算器本身，而且包括执行数据运算所需要的全部控制线路。

从图 4.11 可以看到，80287 内部有处理浮点数指数部分的部件和处理尾数部分的部件，有加速移位操作的移位器线路，它们通过指数总线和小数总线与 8 个 80 位字长的寄存器堆栈相连接。这些寄存器可以按堆栈方式工作，此时，栈顶被用作累加器；也可以按寄存器的编号直接访问任意一个寄存器。8 个寄存器的编号用 0～7 表示，处在栈顶的那个寄存器的编号由 80287 的状态字字段 TOP 给出。在 80287 的指令中，用 ST 表示栈顶寄存器，并且可以用 ST(i)来访问其他 7 个寄存器，此时 i 值可以为 1～7 中的一个值。i 是相对于栈顶的一个偏移量，而不一定是真正 8 个寄存器的实际编号。

3 种浮点数，阶码的基数均为 2，阶码用移码表示，尾数用原码表示。浮点数有 32 位、64 位和 80 位 3 种长度。80287 在从存储器取数和向存储器写数时，均用 80 位的临时实数和其他数据类型执行自动转换。

为了保证指令的正确执行,80287内还设置了各为16位字长的3个寄存器,即特征字寄存器、控制字寄存器和状态字寄存器。

2. Pentium 计算机中配置的运算器

图4.12给出了Pentium计算机中配置的运算器的组成框图。

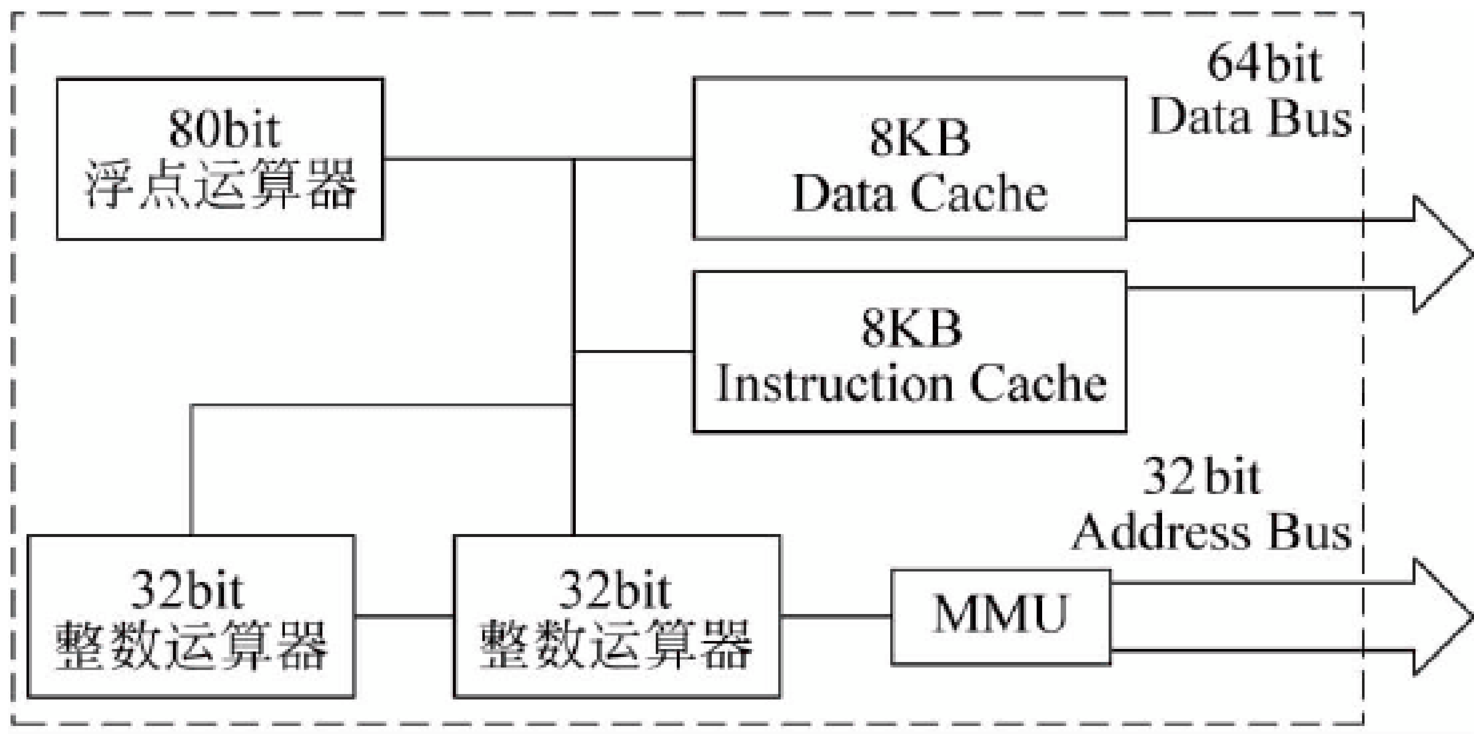


图4.12 Pentium 机的运算器组成示意图

它有2个32位的完成整数运算的定点运算器和一个完成80位的浮点数运算的浮点运算器。仅从这里考虑,它就可以同时执行2条对32位整数运算的指令和1条对80位浮点数运算的指令的运算处理。再考虑到对整型数据运算的指令和对浮点数运算的指令,又都采用了多级的流水线处理方案,事实上就可以有许多条指令同时处在执行过程中,其处理能力比只用一个不采用流水线方案的定点运算器的处理能力要增强许多倍。当然,在流水线方式下的运行效率,并不是总能达到最佳,还与其他某些因素有关,等到讲解指令流水的章节再加以说明。

本章内容小结和学习方法建议

本章主要介绍运算器部件,重点围绕定点运算器的功能、组成与实现来进行讲解。在讲解通用知识的基础上,还给出了一个很典型的定点运算器芯片实例Am2901,较为详细地介绍其内部组成逻辑框图、实现的功能及控制方式,用这种芯片构建一个16位字长的运算器部件的方法。还介绍了MIPS32计算机运算器的组成概况。

浮点数的表示与编码,浮点数的运算规则应该掌握,对浮点运算器的组成只是点到为止,一般了解即可,不必深究。

本章给出了一个原理性的运算器模型,强调学习基础原理知识和了解线路实现,两个运算器实例,在教学过程中它们不是并列关系,只要学懂其中的一个即可。例如用4片Am2901芯片构成的运算器部件即可,而把另外的一个作为开阔眼界的背景资料来阅读,比较它们之间的相同与差异之处,并从几个特例中抽取出普遍适用的原理知识,包括一个CPU系统由运算器和控制器两部分组成,运算器主要由寄存器组(REGs)、ALU、标志位寄存器(Flag)这3个部分(和其他配套电路)组成。而出于不同的设计目标、不同的使用要求,可以在不同的系统中对这3个部分选用不同的处理方案。请注意,把寄存器组作为运算器的一部分,而不是简单地把ALU理解为完整运算器更合理一些,逻辑关系更清晰,也更容易理解。

Am2901是20世纪70年代初的产品,是定制的4位的位片结构的运算器芯片,用作运

算器教学实例具有很好的典型性。它有2个明显特点：①它是比较完整的运算器芯片，包含寄存器组(REGs)和算术逻辑运算单元(ALU)两部分电路，读出REGs中2个寄存器的内容、ALU执行运算、把运算结果写回目的寄存器这3项操作在一个时钟周期完成，难以用于实现指令流水。②内含一个乘商寄存器Q，ALU又支持左、右移位功能，很容易实现一位的硬件乘除法指令，清楚地反映那个时代的集成电路和计算机系统的发展水平。

提醒同学们注意，运算器是计算机系统中最容易学懂、有可能深入理解的功能部件，也是计算机控制器控制起来最为复杂的部分。阅读描述CPU功能与组成的ABEL-HDL或VHDL语言的源程序就能够清楚地看到这种情况。学生在学习计算机组成原理课程的过程中容易遇到的误区之一，就是只看到控制器难学，其实对运算器部件理解不透可能是其根源之一。在完整地学习过CPU系统之后会对这一问题有更清楚的认识。这一小段文字强调学好运算器部件的重要性。

习题与思考题

1. 运算器的主要功能是进行________。
   A. 逻辑运算　　B. 算术运算
   C. 逻辑运算和算术运算　　D. 只作加法
2. 运算器虽由许多部件组成，但核心部分是________。
   A. 数据总线　　B. 算术逻辑运算单元
   C. 多路开关　　D. 累加寄存器
3. 定点运算器的ALU通常可以完成对哪几种数据的运算？
4. 定点运算器一般由哪几部分组成？
5. ALU属于________部件。
   A. 运算器　　B. 控制器　　C. 存储器　　D. 寄存器
6. 简述书中给出的MIPS计算机的运算器部件的功能和组成。
7. 解释定点运算器的功能和组成逻辑部件。以Am2901芯片为实例，来介绍定点运算器的有关知识，你的认可程度如何？
8. 在定点二进制运算器中，减法运算一般通过________来实现。
   A. 原码运算的二进制减法器　　B. 补码运算的二进制减法器
   C. 补码运算的十进制加法器　　D. 补码运算的二进制加法器
9. 加法器采用并行进位的目的是________。
   A. 提高加法器的速度　　B. 快速传递进位信号
   C. 优化加法器结构　　D. 增强加法器功能
10. 组成一个运算器需要多个部件，但下面所列________不是组成运算器的部件。
    A. 状态寄存器　　B. 数据总线　　C. ALU　　D. 地址寄存器
11. 在运算器部件中，为什么要设置多个累加器？累加器的数目多少对计算机的性能有什么影响吗？
12. 浮点运算器由哪几部分组成？
13. 某计算机有一个标志寄存器，其中有进位/借位标志CF、零标志ZF、符号标志SF

和溢出标志 OF,条件转移指令 bgt(无符号整数比较大于时转移)的转移条件是________。

A. CF+OF=1　B. $\overline{\text{SF}}$+ZF=1　C. $\overline{\text{CF+ZF}}$=1　D. $\overline{\text{CF+SF}}$=1

14. 浮点运算器的组成比定点运算器组成更复杂,主要表现在哪些方面?原理上讲,浮点运算器由两个相互关联的定点运算器(分别处理整数和纯小数)组成,有点道理吗?

15. 假定 $X=0.0110011\times2^{11}$,$Y=0.1101101\times2^{-10}$(此处的数均为二进制),再不使用隐藏位的情况下,完成下列题目。

(1) 浮点数阶码用4位移码、尾数用8位原码表示(含符号位),写出该浮点数能表示的绝对值最大、最小的(正数和负数)数值。

(2) 写出 X、Y 的正确的浮点数表示(注意,此处预设了个小陷阱)。

(3) 计算 $X+Y$。

(4) 计算 $X\times Y$。

16. 假定在一个8位字长的计算机中运行如下的类C程序段:

```
unsigned int x=134;
unsigned int y=246;
int m=x;
int n=y;
unsigned int z1=x-y;
unsigned int z2=x+y;
int k1=m-n;
int k2=m+n;
```

若编译器编译时将8个8位寄存器 $R_1\sim R_8$ 分别分配给变量 x、y、m、n、z_1、z_2、k_1、k_2。请回答下列问题。(提示:带符号整数用补码表示)

(1) 执行上述程序段后,寄存器 R_1、R_5 和 R_6 的内容分别是什么?(用十六进制表示)

(2) 执行上述程序段后,变量 m 和 k_1 的值分别是多少?(用十进制表示)

(3) 上述程序段涉及带符号整数加/减、无符号整数加/减运算,这4种运算能否利用同一个加法器及辅助电路实现?简述理由。

(4) 计算机内部如何判断带符号整数加/减运算的结果是否发生溢出?上述程序段中,哪些带符号整数运算语句的执行结果会发生溢出?

17. 一个C语言程序在一台32位机器上运行。程序中定义了3个变量 x、y 和 z,其中 x 和 z 为 int 型,y 为 short 型。当 $x=127$,$y=-9$ 时,执行赋值语句 $z=x+y$ 后,x、y 和 z 的值分别是________。

A. x=0000007FH, y=FFF9H, z=00000076H

B. x=0000007FH, y=FFF9H, z=FFFF0076H

C. x=0000007FH, y=FFF7H, z=FFFF0076H

D. x=0000007FH, y=FFF7H, z=00000076H

18. 浮点数加、减运算过程一般包括对阶、尾数运算、规格化、输入和判溢出等步骤。设浮点数的阶码和尾数均用补码表示,且位数分别为5位和7位(均含2位符号位)。若有两个数 $X=2^7\times29/32$,$Y=2^5\times5/8$,则用浮点加法计算 $X+Y$ 的最终结果是________。

A. 00111　1100010　B. 00111　0100010

C. 01000　0010001　　　　D. 发生溢出

19. 假定变量 i、f 和 d 的数据类型分别为 int、float 和 double(int 用补码表示，float 和 double 分别用 IEEE 754 单精度和双精度浮点数据格式表示)，已知 $i=785$，$f=1.5678e^3$，$d=1.5\ e^{100}$。若在 32 位机器中执行下列关系表达式，则结果为"真"的是________。

(Ⅰ)i==(int)(float)i　　　　(Ⅱ)f==(float)(int)f

(Ⅲ)f==(float)(double)f　　　　(Ⅳ)(d+f)-d==f

A. 仅Ⅰ和Ⅱ　　B. 仅Ⅰ和Ⅲ　　C. 仅Ⅱ和Ⅲ　　D. 仅Ⅲ和Ⅳ

20. 下列选项中，描述浮点数操作速度指标的是________。

A. MIPS　　B. CPI　　C. IPC　　D. MFLOPS

21. float 型数据通常用 IEEE 754 单精度浮点数格式表示。若编译器将 float 型变量 x 分配在一个 32 位浮点寄存器 FR1 中，且 $x=-8.25$，则 FR1 的内容是________。

A. C104 0000H　　B. C242 0000H　　C. C184 0000H　　D. C1C2 0000H

第5章 指令系统和汇编语言程序设计

指令是指示计算机执行某项运算或操作功能的命令。一台计算机使用的全部指令组成这台计算机的指令系统。指令系统处于硬件和软件的交界面上，它被划分为精简指令系统(RISC)和复杂指令系统(CISC)两大类。从用户的角度看，指令用于编写程序，是用户使用与控制计算机运行的最小功能单位；从计算机的组成和功能来看，计算机硬件系统是用于实现每条指令功能的实际设备，因此硬件系统只能直接识别和运行由机器指令构成的程序，指令系统直接与计算机的运行性能和硬件结构密切相关，是设计一台计算机的起始点和基本依据。

本章重点讲授计算机的指令、指令系统和汇编语言程序设计这3项内容。

5.1 指令格式和指令系统概述

5.1.1 指令的定义和指令格式

计算机系统由硬件和软件两个子系统组成。硬件是指构成计算机的中央处理机、主存储器、输入输出设备等物理装置。软件则指由软件厂家为方便用户使用计算机而提供的系统软件和用户用于完成自己的特定事务和信息处理任务而设计的用户软件。计算机硬件能直接识别和运行的软件程序通常由该计算机的指令代码序列组成。

1. 指令的定义

用于组成计算机程序、指示计算机硬件执行某项运算或操作功能的命令叫作指令，在计算机内部它是用一定长度的二进制位串来表示的。一台计算机支持(或称使用)的全部指令构成该机的指令系统。指令系统对计算机用户和计算机厂家都有着非常重要的影响。指令是设计计算机程序的最小功能单位，计算机厂家需要在计算机硬件系统中实现每一条指令的功能，用指令设计各种系统软件，指令系统直接与计算机系统的运行性能、硬件结构的复杂程度等密切相关，它是设计一台计算机的起始点和基本依据。

早期的计算机，从简化计算机硬件结构、降低成本考虑，指令系统都比较简单，指令条数少、运算功能弱，能处理的数据只是定点小数，使用相当困难。到了20世纪六七十年代，随着集成电路和超大规模集成电路的出现与发展，计算机硬件成本直线下降，相应的软件成本所占比例迅速增加，计算机的指令系统日渐变得更加复杂和完备，指令条数多达300～500条，寻址方式也多达十几种，能直接处理的数据类型更多，构成了复杂指令系统的计算机。

在1975年前后,人们又发现,一味追求指令系统的复杂和完备程度,并不是提高计算机性能的唯一途径。在CISC计算机中,有80%的功能更强、实现起来更为复杂的指令却较少被使用,在程序运行的过程中只占到20%的时间,有80%的程序运行时间使用的是另外20%的功能简单、实现容易的指令。据此提出了简化指令系统的计算机的概念并予以实现,只选用几种简单的寻址方式和最常用的几十条指令,充分考虑了超大规模集成电路设计、制造中的有关问题,吸收当前软件研究的各项成果,从硬、软件结合的角度解决了许多矛盾,设计制造出运行性能更高的RISC计算机系统。它虽然有明显的优势,但不能完全取代CISC计算机系统,考虑到此前已有的大量软件资源是应用CISC结构的指令系统实现的,CISC计算机仍有其存在的理由。

要确定一台计算机的指令系统并评价其优劣,通常应从如下4个方面考虑。

(1) 指令系统的完备性,常用指令齐全,编程方便。

(2) 指令系统的高效性,程序占内存空间少,运行速度快。

(3) 指令系统的规整性,指令和数据使用规则统一简单,易学易记。

(4) 指令系统的兼容性,同一系列的低档计算机的程序能在新的高档机上直接运行。

要完全同时满足上述标准是困难的,但它可以指导我们设计出更加合理的指令系统。设计指令系统的核心问题是选定指令的功能和格式。指令的格式与计算机的字长、期望的存储器容量和读写方式、支持的数据类型、计算机硬件结构的复杂程度和追求的运算性能等有关,这些内容中的某些部分超出了我们的教学内容范围,请有兴趣者参阅其他资料。

2. 指令格式

通常情况下,一条指令要由如图5.1所示的两部分内容组成。

操作码	操作数地址

图5.1 一条指令的组成

第一部分是指令的操作码。操作码用于指明本条指令的运算和操作功能,例如,是算术加运算、减运算还是逻辑与运算、或运算功能,是否是读、写内存或读、写外设操作功能,是否是程序转移和子程序调用或返回操作功能等。计算机需要为每条指令分配一个确定的操作码。

第二部分是指令的操作数地址。用于给出被操作的信息(指令或数据)的地址,包括参加运算的一个或多个操作数所在的地址,运算结果的保存地址,程序的转移地址,被调用的子程序的入口地址等。

在一条指令中,如何安排指令字的长度,即使用多少个二进制位(bit)表示一条指令,又如何分配指令字中这两部分所占用的位数(长度),如何安排操作数的个数,如何表示和使用一个操作数的地址(寻址方式),是要认真对待、精心设计的重要问题。寻址方式将单独放在5.2节讲解,其余内容在本节的下面部分分别介绍。

5.1.2 操作码的组织与编码

计算机的字长是由选用的数据的类型及其表示所用的二进制位数决定的,通常是2、4、8个字节。指令字的长度,多数情况下就确定为计算机的字长,即一条指令占用计算机的一个内存字,但并不强制要求所有指令的字长都相同,以便提高计算机资源利用率。例如,在一个内存字中,可以考虑存放几条很短的指令;长的指令也可能占用多个内存字,例如在字长较短的计算机系统中,某些指令需要选用双字指令格式,保存这样的一条指令就要使用两

个内存字。

从对指令操作码的组织与编码所选用的方案分类,可以区分出如下2种处理情况。这里只是简单介绍一些基本概念,更详细的内容到5.3节会清楚地看到。

(1) 定长的操作码的组织方案。在当前多数的计算机中,一般都在指令字的最高位部分分配固定的若干位(定长)用于表示操作码,例如8位,它有256个编码状态,故最多可以表示256条指令。这对于简化计算机硬件设计,提高指令译码和识别速度很有利。当计算机字长为32位或更长时,这是常规正统用法,如IBM370机、VAX-11机。我们设计实现用于硬件课程教学的16位字长的教学计算机系统也选用定长的8位指令操作码。

(2) 变长的操作码的组织方案。当计算机的字长与指令长度为16位或8位时,单独为操作码划分出固定的多位后,留给用于表示操作数地址的位数就会严重不足。为此不得不对一个指令字的每一个二进制位的使用精打细算,使一些位(bit)在不同的指令中有不同的作用。例如,在一些指令中,这些位用作操作码;而在另外一些指令中,这些位又被用作操作数的地址。则不同指令的操作码长度就会不同,即尽量为那些最常用(程序中使用频率高)、用于表示操作数地址的位数要求又较多的指令,适当少分配几位操作码(当然能表示的指令条数也就少);而对那些表示操作数地址的位数要求较少的指令多分配几位操作码;对那些无操作数的指令,整个指令字的全部位都用作操作码,力争在比较短的一个指令字中,既能表示出比较多的指令条数,又能尽量满足给出相应的操作数地址的要求,如PDP-11计算机就选用这种方案。我们设计实现的用于硬件课程教学的8位字长的计算机就选用变长的操作码。

5.1.3 有关操作数的类型、个数、来源、去向和地址安排

在计算机指令中,可以直接使用的基本数据类型通常包括逻辑类型(bit),字符和字符串类型、整数类型(integer)、浮点数类型(floating)等,需要结合指令的操作码来判断数据类型并完成相应的运算处理。使用这几个基本的数据类型还能构造出高级语言中的更为复杂的复合数据类型,这些复合数据类型不能出现在基本的机器指令中。

不同的指令使用不同数目、不同来源去向、不同用法的操作数,必须把它们统一起来,并安排在指令字的数据地址字段。

从用到的操作数个数区分,可能有如下4种情况。

(1) 无操作数指令。有的指令不涉及操作数,或使用约定的某个(些)操作数,既已约定则没有必要再在指令中加以表示,称这类指令为无操作数指令。它仅有操作码部分,例如停机指令、空操作指令、关中断指令、堆栈结构的计算机系统中对堆栈中数据运算的指令等。

(2) 单操作数指令。有些指令只用一个操作数,必须在指令中指明其地址,如一个寄存器内容增1或减1运算的指令;或还使用约定的某个操作数,既已约定则无须再在指令中加以表示,例如完成从(向)外设读(写)数据的指令,就可以只在指令中指明该外设地址,而把接收(送出)数据的通用寄存器约定下来。此外,在短字长的、采用单个累加器的计算机中,已约定目的操作数(如被加数、被减数等)和保存计算结果都使用唯一的那个累加器,指令中只需指定另外一个源操作数即可。称上述这些指令为单操作数指令。

(3) 双操作数指令。对于常用的算术和逻辑运算指令,往往要求使用两个操作数,这两个操作数往往保存在两个寄存器中,需分别给出目的寄存器和源寄存器的编号,其中目的寄存器还用于保存本次的运算结果。在寄存器与主存储器之间完成数据传送的指令,也需要

给出寄存器编号和主存储器地址。称这类指令为双操作数指令。

(4) **多操作数指令**。另外一些指令可能使用多个操作数,如3个操作数,其中两个操作数地址分别给出两个源操作数的地址,第三个操作数地址是用于给出保存本次运算结果的目的操作数地址,可以称这类指令为三操作数指令;在有些性能更高的计算机(甚至PC机)中,还有在指令中使用更多个操作数地址的指令,用于完成对一批数据的处理过程,如字符串复制指令,向量、矩阵运算指令等,称这类指令为多操作数指令。

上述4种情况中的前3种,由于其具有指令字长可以相对较短、执行速度较高、计算机硬件结构可以相对简单等优点,在各种不同类型的计算机中被广泛应用;相对而言,最后一种更多地用在字长较长的大中型计算机中。

请注意,指令的操作数地址字段有时也用于给出指令的地址,例如,转移指令的转移目标地址、子程序的入口地址等。

在设计实现实际的指令系统时,还有一些问题需要处理,暂不赘述。

从操作数的来源、去向及其在指令字中的地址安排考虑,有多种情况。

这里说的操作数的来源、去向,是指表示在指令中操作数要从哪里读来、写向哪里去。

下面讨论操作数的来源、去处和地址安排。

可以把指令中的操作数的来源、去向归纳为如下4种主要情况。

(1) 操作数的第一个来源、去向,可以是CPU内部的通用寄存器,此时应在指令字中给出用到的寄存器编号(寄存器名)。通用寄存器的数量一般为几个、十几个,一二百个,故在指令字中须为其分配2、3、4、5或更多一点的位数来指明一个寄存器。该寄存器中的内容,可以是指令运算用到的数据,也可能用作一个操作数或者指令的地址,或计算主存储器地址的相关信息。

(2) 操作数第二个来源、去向,是外围设备(接口)中的一个寄存器。通常用设备编号、或设备入出端口地址、或设备映像地址(与内存储器地址统一编址的一个设备地址编号)来表示。设备编号或设备入出端口地址用的位数不会太多,通常可以在第一个指令字中直接给出。设备映像地址与一个内存单元地址的处理办法类同,到5.2节具体说明。

(3) 操作数的第三个来源、去向,是内存储器的一个存储单元,此时应在指令字中给出该存储单元的地址。由于许多计算机用到的内存地址的位数就是一个机器字的长度,要在一个指令字中既给出操作码、有关寄存器编号等信息,又直接给出这一内存地址是困难的,再加上还有采用多种不同方式读写内存的需求,这就需要找到解决这些问题的完整方案。将单独在5.2节来详细说明具体解决方案。

(4) 操作数的第四个来源是指令字中的某个字段的内容。它可能就是一个运算要用到的数据,或适当处理后形成用于数据运算、存储器地址计算的一个数据。

5.1.4 指令的分类

一台计算机的指令系统往往由几十条到几百条指令组成。可以从不同的角度对这些指令进行分类。如前一节讲的就是按指令中使用的操作数个数来分类的,但更常用的还是按指令所完成的功能进行分类,由于不同人对指令功能划分的标准并不完全相同,其分类结果也就会有所差异,我们给出如下分类的例子。

(1) 算术与逻辑运算类指令是每台计算机都必须具有的指令,它通常完成对一或两个

数据的算术运算或逻辑运算功能。不同档次、不同性能的计算机所能加工、运算的数据类型(整数、浮点数、十进制数等)和支持的运算功能(加、减、乘、除、变符号等)的完备程度可以有所差异,但一般都必须给出算术与逻辑运算的结果,以及结果的有关特征。

(2) 移位操作类指令包括算术移位、逻辑移位、循环移位3种,用于把指定的一个操作数左移或右移一(多)位。从实用的角度,算术移位指令常只使用右移功能,对补码表示的一个二进制数,最高位上的符号位不变,再把符号位和数据位同时右移一位;逻辑左(右)移位通常是在最低(高)的一位移入0值,把最高(低)一位移出的值送到进位触发器C中;循环移位则是把进位触发器C和一个被移位的数据首尾衔接起来一同实现左或右移位操作。

(3) 数据传送类指令用于实现通用寄存器之间、通用寄存器与内存储器存储单元之间(通常也可以单独称其为存储器读写指令)、内存储器不同的存储单元之间、通用寄存器与外围设备(接口)之间(有些场合也可以单独划分为输入/输出指令)的数据传送功能。从内存储器和外围设备(接口)操作性质的不同,又可以区分为读和写操作两种,它指明数据传送的方向,数据从内存或外围设备传送到CPU,或者数据从CPU传送到内存或外围设备。

(4) 转移类指令用于解决变动程序中指令执行次序(程序执行流程)的需求。转移指令是一种改变程序中指令执行次序的指令。程序中的指令,大部分是按指令排列的自然次序顺序执行,但有时候又要求不执行邻接的下一条指令,而是转移到另一段程序入口处开始执行,此时就会用到一条转移指令。转移指令又被区分为无条件转移和条件转移两类。二者的相同之处是都必须在指令中给出转移地址。不同之处是,条件转移指令还必须在指令中给出判断是否执行转移所依据的条件,仅在条件为真时才执行转移,否则顺序执行;而无条件转移指令一定执行转移。

(5) 子程序调用指令与返回指令可以被理解为一种特定的转移指令。二者要配合使用,通过子程序调用指令,使一段被称为子程序的特定程序段投入运行,而该程序段的最后一条指令,一定是一条子程序返回指令,它会在子程序运行结束后,确保转移回到主程序中排在子程序调用指令之后的那条指令处接着执行。而一般的转移指令,并不涉及再返回来的问题。再深入一步讲,子程序又可以被分为用户自己编写的子程序和软件系统提供的子程序两大类,这后一部分又被称为访问系统程序(访管)指令、陷阱(TRAP)指令。

(6) 特权指令是指仅用于操作系统或其他系统软件的指令,为确保系统与数据安全起见,这一类指令不提供给用户使用。这一类指令主要用于管理与分配系统资源,包括改变系统的工作方式,完成任务的创建或切换,变更管理存储器用的段表和页表中的内容等。特权指令对多用户或多任务系统是必要的。

(7) 其他指令,如动态停机指令、空操作指令、置条件码指令、开中断指令、关中断指令、堆栈操作指令等,用于完成某些特定的处理功能。

5.1.5 指令周期及其对计算机性能和硬件结构的影响

本节内容要到第6章控制器部件的相关章节详细讲解,先在这里简单介绍有关指令周期和指令执行步骤两个基本概念。

通常把执行一条指令所用的时间叫作指令周期。一个指令周期往往要包含几个执行步骤,就是说,一条指令的完整功能需要依次经过这几个步骤的操作才能完成。例如在MIPS计算机系统中,一个指令周期可能包括读取指令、指令译码和读寄存器组、ALU执行运算、

读写内存或接口、数据写回寄存器组这5个步骤，如图5.2所示。



图5.2　指令的5个执行步骤

所有指令都是在取指—译码—执行(包括第(3)～(5)步)的循环中完成的。

(1) 读取指令是每一条指令必须首先完成的，所完成的功能对所有指令都相同。

(2) 指令译码完成的功能对多数的指令是类似的，例如判断指令类型、读寄存器组等。

(3) ALU执行运算所完成的是数据计算或者地址计算功能，对不同指令会有所差别。

(4) 读写内存或接口仅出现在读写内存或者读写接口指令的执行过程中。

(5) 数据写回完成把ALU的计算结果或从内存、接口读来的数据写入寄存器组。

需要注意，这5个执行步骤的前后次序是有内在联系的，每一步都会作为下一步运行的前提，不能随意变动。第1步的读取指令是后续各步骤运行的前提，只有在取得指令之后才能够知晓接下来应该完成的是什么运算处理功能、使用哪几个数据、需要怎样完成运算处理等；在第1步中还可以计算出下条相邻指令所在的存储单元地址。

在得到指令内容之后，第2步就可以检查、判别指令类型、指令操作码等。这主要体现为对指令的操作码字段内容进行译码；大部分指令执行过程中会用到寄存器组内的1～2个寄存器的内容，要在此步骤把它们读出来，作为下一步ALU执行计算的原始数据。

第3步是ALU执行运算，对于寄存器之间的数据运算等指令，完成的是对数据的运算、比较、移位等操作。运算结果通常需要写回到寄存器组的一个寄存器中，在这里已把写回操作移到更后面的一个步骤中去完成，在本步骤将把计算结果暂存到一个专用寄存器中。对存储器读写指令，ALU完成的是计算数据在存储器中的单元地址，而真正的存储器读写操作需要放到再下一个步骤中去完成。对转移等指令，ALU完成的是计算转移指令的目标地址。

第4步用于完成存储器读写功能，使用第3步计算得到的存储器地址选中的一个存储单元完成数据读写。写内存指令用于把指定的一个寄存器的内容写入选定的存储单元，读内存指令用于把选定的存储单元的内容读出来并暂存在一个专用的寄存器中，而真正写入到通用寄存器中的操作需要放到再下一个步骤中去完成。

第5步用于把第3步ALU运算的数据结果(已暂存在一个专用寄存器中)或者第4步从存储器中读出来的数据(已暂存在一个专用寄存器中)写入到指令指定的通用寄存器中。

综上所述，数据运算指令需要依次经过第1、2、3、5这4个步骤完成，写存储器指令需要依次经过第1～4这4个步骤完成，而读存储器指令需要依次经过第1～5这5个步骤完成。另外一些指令也可能经过第1～2这2个步骤或者经过第1～3这3个步骤完成。

初步结论，并不是所有的计算机都必须把一条指令的执行过程划分为这样的5个步骤，也不见得每一条指令都必须经过全部这5个步骤才能完成。例如，功能简单的指令可能只用到前第2和第3个步骤，另外一些指令可能还用到后面的第4和第5个步骤。在计算机内部，从如何选择、处理这5个执行步骤的角度来看，有以下3种方案，每个方案对指令执行速度和计算机硬件实现的复杂程度等有较大影响。

第1种方案，不管指令功能的复杂程度，让所有指令都经过这5个时间段完成，即全部指令都选用同一种指令周期，从而构成单指令周期的CPU系统。此时必须依照功能最复

杂、执行用时最长的指令的运行要求来确定指令周期的长度,否则系统运行会出错。对那些功能简单,只用更少执行步骤就能完成的指令,势必遇到一些无操作功能的执行步骤,造成执行时间上的浪费,降低了系统的运行效率。此外,这个方案不利于在一条指令执行过程中的硬件部件的分时共享,系统资源利用率最低。结论是:这种方案理论上可行,现实中不实用。

第 2 种方案,依据不同指令功能的繁简程度来分别确定各自的指令周期,指令需要几个执行步骤完成就为它分配几个步骤,此时可能有分别用 2 个步骤、3 个步骤、4 个步骤或 5 个步骤完成的指令,此而构成多指令周期的 CPU 系统。这一方案明显优于前一方案,指令执行速度和硬件部件的资源利用率更高,是一种较为实用的方案。

第 3 种方案,属于指令流水线的 CPU 系统,它的着眼点是进一步提高指令的执行速度。思路在于尽可能地在指令的每一个执行步骤都启动一条新指令,使程序中几条相邻指令的执行时间尽可能地重叠起来,就像工业生产中的流水线,故称指令流水线。它与前 2 种方案有明显区别。前 2 种方案实现的是指令的串行执行,即前一条指令完成之前不会启动下一条指令,在任何时间都只有一条指令处在运行中,而在指令流水的 CPU 系统中,当确定采用 5 个步骤的指令流水线时,理想情况可能有 5 条指令各自处在 5 个不同的步骤中,其理想情况可以做到每一个执行步骤都会有一条指令结束执行过程。这种方案会使得硬件设计与实现更复杂一些,但带来了更高的性能价格比,是目前被广泛选用的、最为实用的方案。

5.2 基本寻址方式概述

寻址方式解决的是如何在指令中表示一个操作数的地址和指令地址,如何用这种表示得到操作数或怎样计算出操作数的地址。表示在指令中的操作数地址,通常被称为形式地址;用这种形式地址并结合某些规则,可以计算出操作数在存储器中的存储单元地址,这一地址被称为数据的物理(有效)地址。计算机中常用的基本寻址方式有以下 8 种类型。

1. 立即数寻址

操作数直接给出在指令字中,即指令字中直接给出的不再是操作数地址,而是操作数本身。它的主要用法是把一个确定的数值传送到一个通用的寄存器中,或直接用于运算。当该数据占用的位数较少时(如小的整数、一个西文字符),可把该数值安排在第一个指令字中,则在读出指令的同时也得到相关数据;否则只能将其存放在指令的第二个字中,这就构成了双字指令,如图 5.3 所示。双字指令不利于指令流水,很少出现在 RISC 计算机的指令系统中。

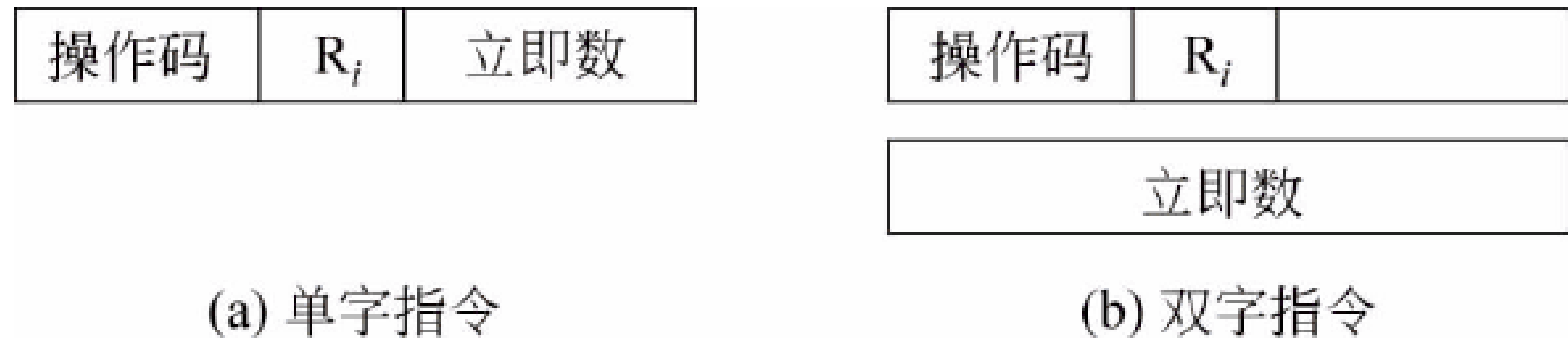


图 5.3 立即数寻址

2. 直接寻址

直接寻址是在指令中直接给出操作数在存储器中的地址。这是计算机中可用的寻址方

式之一。与立即数寻址方式类似，当该地址占用的位数较少时（能访问的存储空间范围较小），可将其安排在第一个指令字中，则在读出指令的同时也得到相关地址，否则只能用双字指令实现，如图5.4所示。更多情况下采用的是变址等其他寻址方式提供内存单元地址。

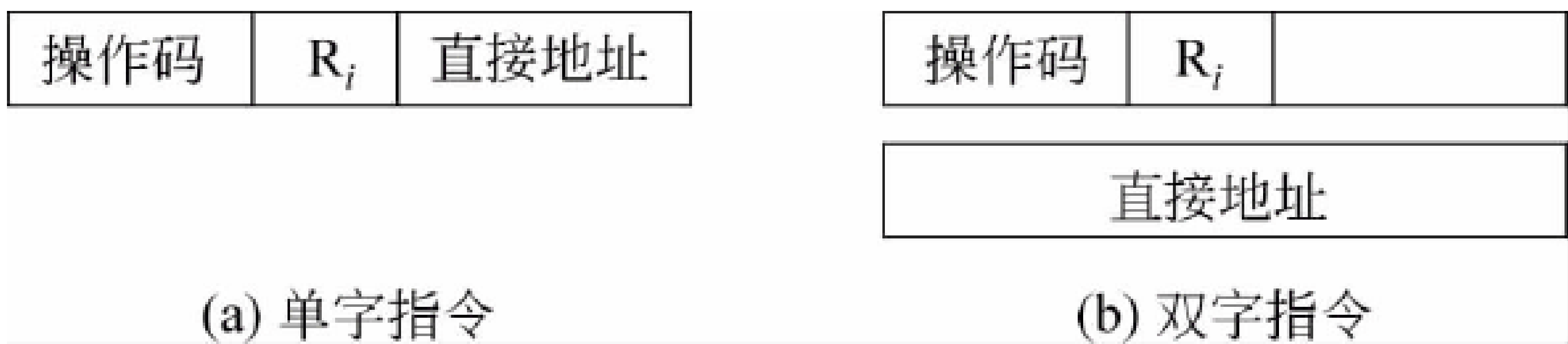


图5.4 直接寻址

3. 寄存器寻址、寄存器间接寻址

寄存器寻址是在指令字中给出通用寄存器的编号（名字、地址），所访问的寄存器的内容就是运算用到的数据。由于表示一个通用寄存器编号占用的位数少，有利于缩短指令字的长度；用寄存器暂存数据并用于完成运算速度更快，故这是最基本最常用的寻址方式。

寄存器间接寻址，在寄存器中给出的不是操作数，而是操作数在存储器中的地址，这被称为寄存器间接寻址，这也是最常用的寻址方式之一，如图5.5所示。

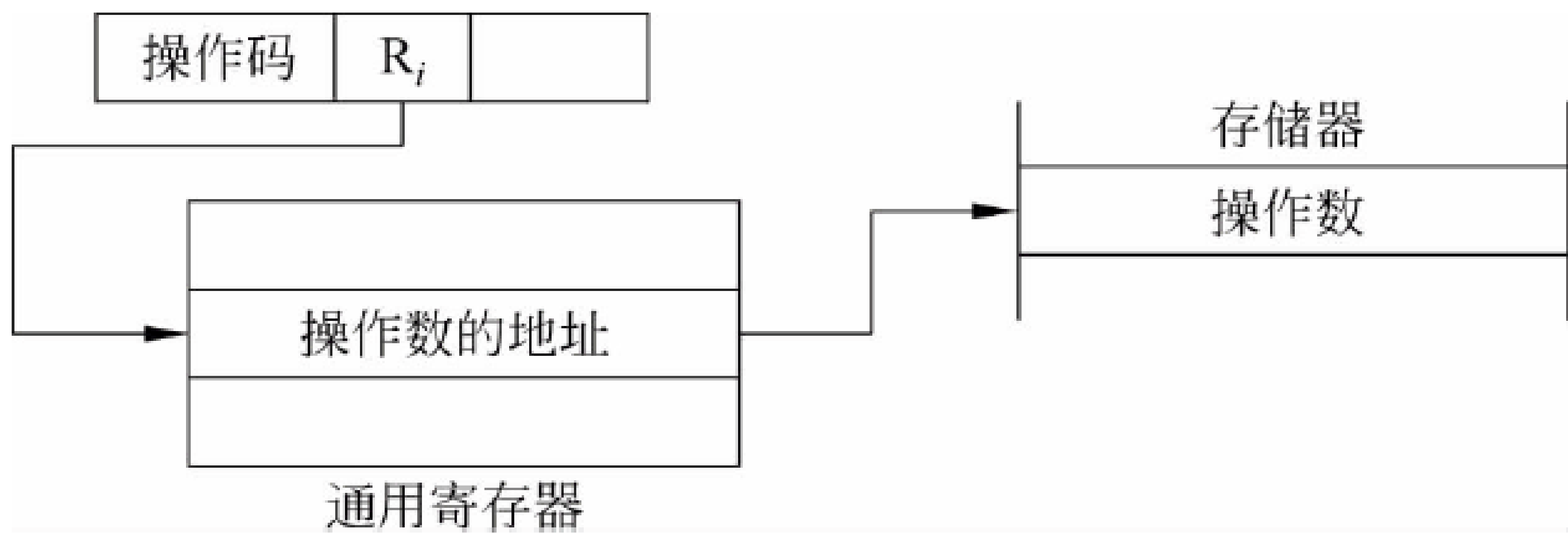


图5.5 寄存器间接寻址

4. 变址寻址

变址寻址是把在指令字中给出的一个数值（称为变址偏移量）与一个被称为变址寄存器的内容相加之和作为操作数的地址，用于读写存储器，如图5.6所示。它特别适合于处理数组型数据。有些计算机，更设置了自动对变址寄存器内容自动增1和减1的操作功能。与立即数寻址方式类似，依据变址偏移量的范围大小，变址指令可能为单字或双字两种情况。

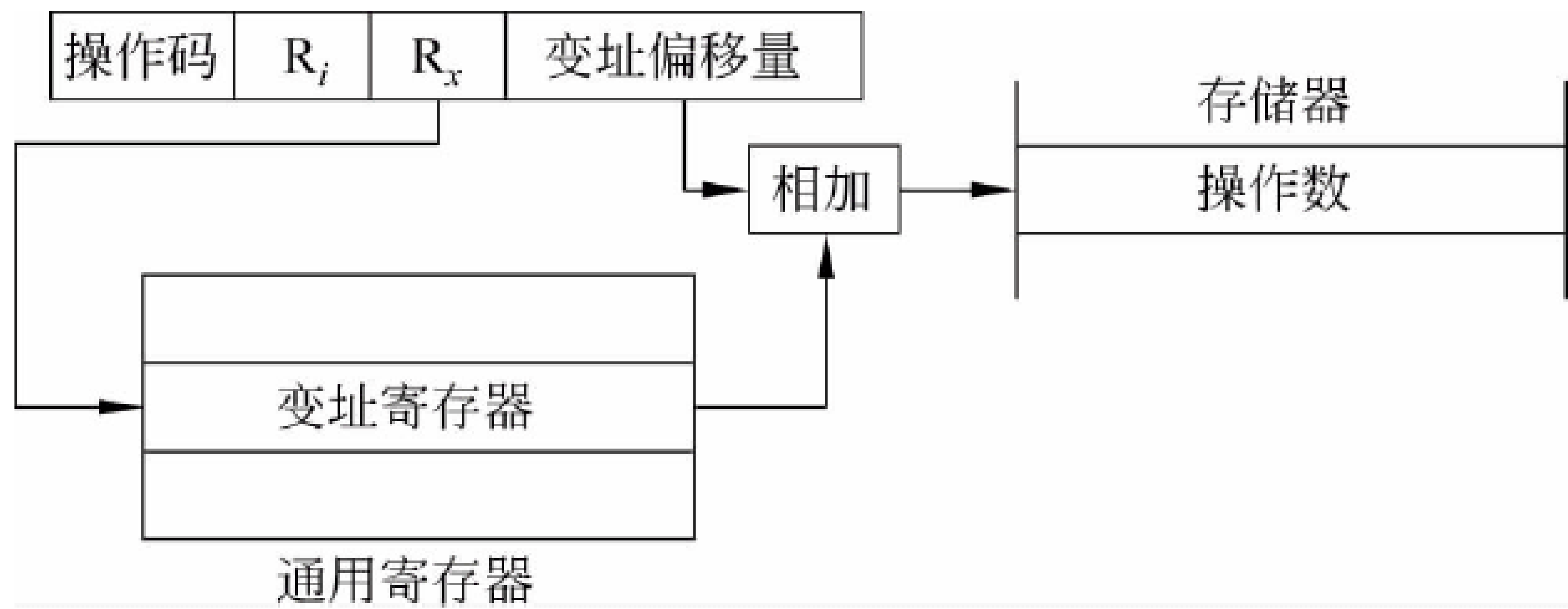


图5.6 变址寻址

5. 相对寻址

相对寻址是指把在指令字中给出的一个数值（称为相对寻址偏移量）与程序计数器PC的内容相加之和作为操作数的地址或转移指令的转移地址，如图5.7所示。相对寻址偏移量决定目标地址与当前指令的地址距离是多少，相对寻址偏移量可以为正值或负值，可以有不同的

取值范围,可能只占一个指令字的一部分(一个字段),其转移的地址范围小,或单独占用一个计算机字,则可以转移到存储器的任何位置,故相对寻址指令也有单字与双字之分。

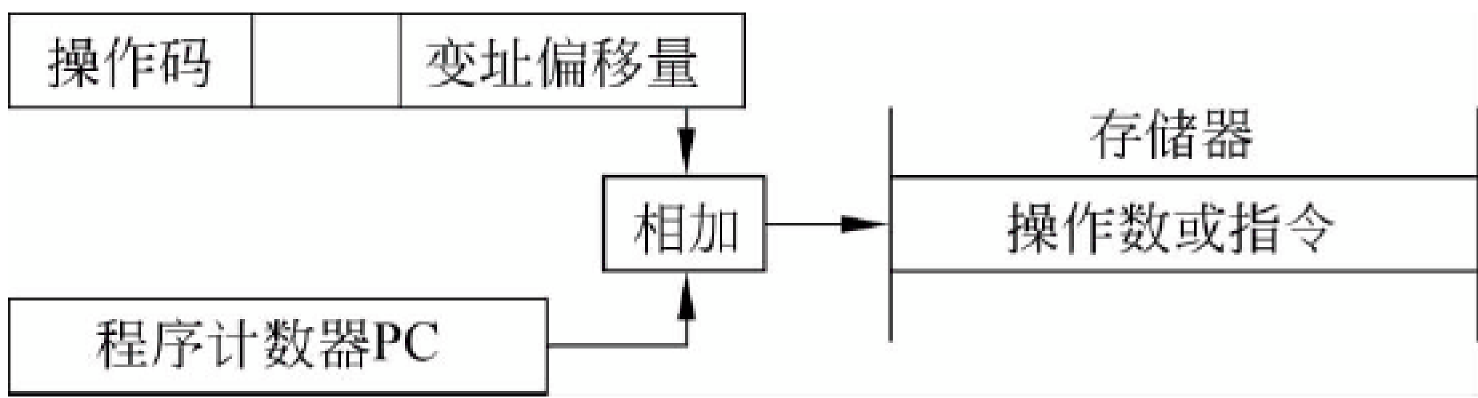


图 5.7　相对寻址

6. 基地址寻址

基地址寻址是指把在程序中所用的地址与一个特定的寄存器(称为基地址寄存器)的内容相加之和作为操作数的地址或指令的地址。它与变址寻址、相对寻址形式上有某些类似之处,但其用法却与二者有很大差别,主要用于为多道程序或浮动地址程序定位存储器空间。

7. 间接寻址

间接寻址,在指令字中给出的不是一个操作数的地址,而是一个操作数地址的地址,或一条指令地址的地址。采用间接寻址读写数据需两次访问存储器,速度较慢,少用为好,如图 5.8 所示。

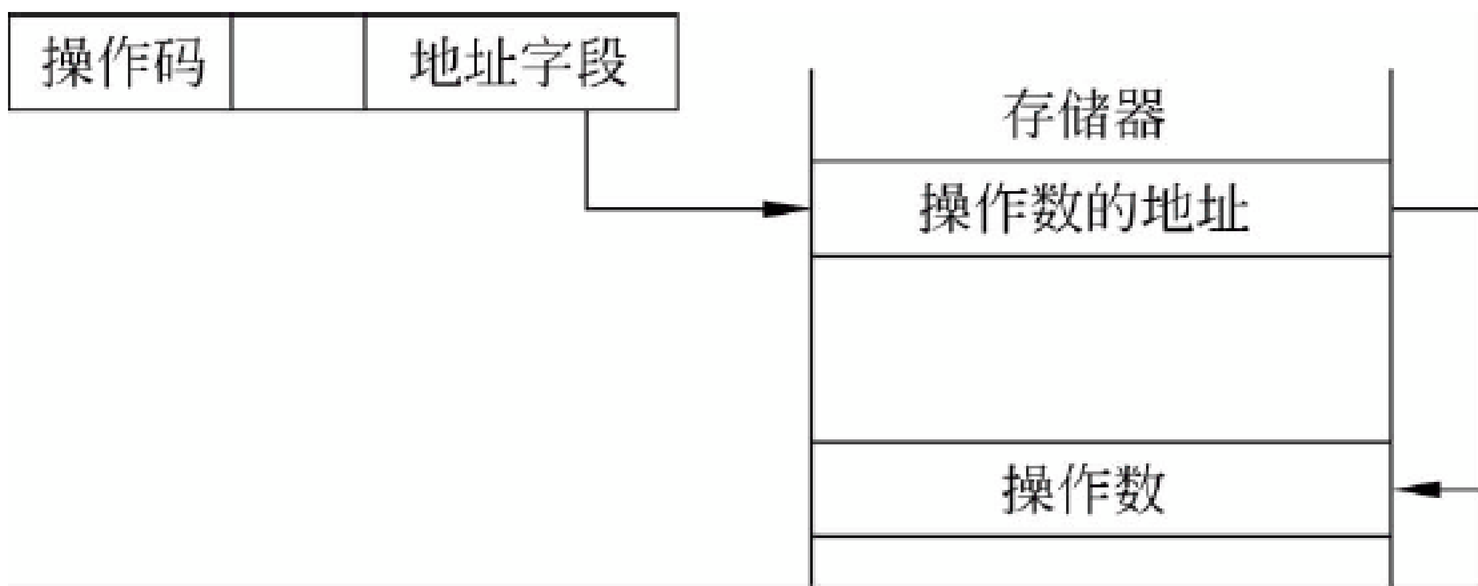


图 5.8　间接寻址

间接寻址的形式地址可以有多种不同方案,图 5.8 中所给出的只是一个最简单的例子。

8. 堆栈寻址

堆栈是存储器中(或专用寄存器组)一块特定的按“后进先出”原则管理的存储区。该存储区中被读写单元的地址是用一个特定的寄存器给出的,该寄存器被称为堆栈指针(Stack Pointer,SP)。如果有些指令,其操作码部分已经指明一个操作数为堆栈中的一个单元(通常为堆栈栈顶)的内容,则它就已经约定将使用 SP 访问该单元,故不必在指令的地址字段中另加指定。在采用堆栈结构的计算机系统中,大部分指令表面上都表现为无操作数指令的形式。通常情况下,在读写堆栈中的一个单元的前后,都伴有自动完成对 SP 内容的增量或减量操作。

请注意,操作数地址字段也被用于给出指令地址,例如转移指令的转移目标地址,子程序的入口地址等。

上述 8 种寻址方式,是计算机中常用的基本寻址方式,可以单独使用,也可以把它们中的某几种组合在一起,如变址后再间接寻址,变址与基地址寻址的组合,间接寻址还可以连续多次执行等。此外,不是每一台计算机都使用所有的寻址方式,也不一定要把寻址方式设

计得很复杂，简单的寻址方式可以使计算机系统有更高的运行效率。

5.3 指令系统举例

计算机的指令系统有 RISC 和 CISC 两种类型。RISC 是 20 世纪 80 年代出现的全新概念的计算机系统。它以精简高效的指令系统、精巧高速的硬件组成、精妙智能的编译软件，达到了低价与高性能的理想目标，也就是说，执行同样处理功能的程序所占用的时间要比 CISC 计算机更短。请看公式：$P=I\times CPI\times T$，符号 P 表示执行一个程序所用的时间，它由该程序中包含的机器指令的总条数 I，执行一机器指令所需要的机器周期数 CPI，每个机器周期的时间长度 T 这 3 个数值的乘积决定。这 3 个数值在 RISC 和 CISC 两种机器中有比较大的差距，如表 5.1 所示，RISC 机器的运行性能可能要比 CISC 机器高 2～5 倍。

表 5.1　RISC 和 CISC 的对比

	I	CPI	T		I	CPI	T
RISC	1.2～1.4	1.3～1.7	<1	CISC	1	4～10	1

RISC 计算机的指令格式规范且种类少，使用的寻址方式简单，指令条数少，指令完成的操作功能简单。大量的统计结果表明，在 CISC 机器的所有指令中，功能简单、所用硬件更节省的约 20%的指令，将占用程序 80%的运行时间；反过来说，另外的约 80%的功能更复杂、硬件实现代价很高的指令并不被经常使用。到了 RISC 计算机中，更倾向选用软件子程序来实现这些指令的功能，使硬件实现变得更为精简，运行速度更快。RISC 计算机追求的目标之一，就是使指令每一步操作所用的时间要尽可能短，并且力争在每个执行步骤都能完成一条指令的执行过程。同时尽力在编译程序中增强性能优化能力，从硬件软件两个方面来提高 RISC 计算机的性能。

下面介绍 4 种计算机的指令系统实例。在这几个指令系统中，指令操作码部分的组织和编码，操作数地址字段的设计方案各有一定的代表性。其中 Pentium Ⅱ 计算机的指令系统属于典型的 CISC 结构；MIPS32 计算机的指令系统属于典型的 RISC 结构；PDP-11 计算机的指令格式，长度扩展的指令操作码和规范、别致的寻址方式是它的突出特点；上述 3 种指令系统更多地用于开阔眼界，扩展学生的知识面。教学计算机的指令系统是由本书作者专门为计算机组成原理课程的教学和实验而设计的，指令格式简明规范，既有多种基本寻址方式、比较齐全的常用指令，又有配套的汇编语言支持，这套指令与多数人较为熟悉的 Intel 8086 计算机的指令有某些类似但更为简化，选用这套指令系统设计、实现了多个型号的教学实验设备，在本教材中说到的教学计算机更多地针对 TEC-XP-Ⅱ 这个型号。教学计算机的指令系统将在教学的各个环节中反复用到，更多地关注一些是必要的。

5.3.1 Pentium Ⅱ 计算机的指令系统

Pentium 计算机是应用面较广、影响力较大的个人计算机，其指令系统有一定的典型性。这里更多地介绍它的指令格式和选用的寻址方式，给出部分指令而不是完整的指令集。Pentium Ⅱ 计算机的指令格式如图 5.9 所示。

下面给出指令前缀各部分的功能简介。

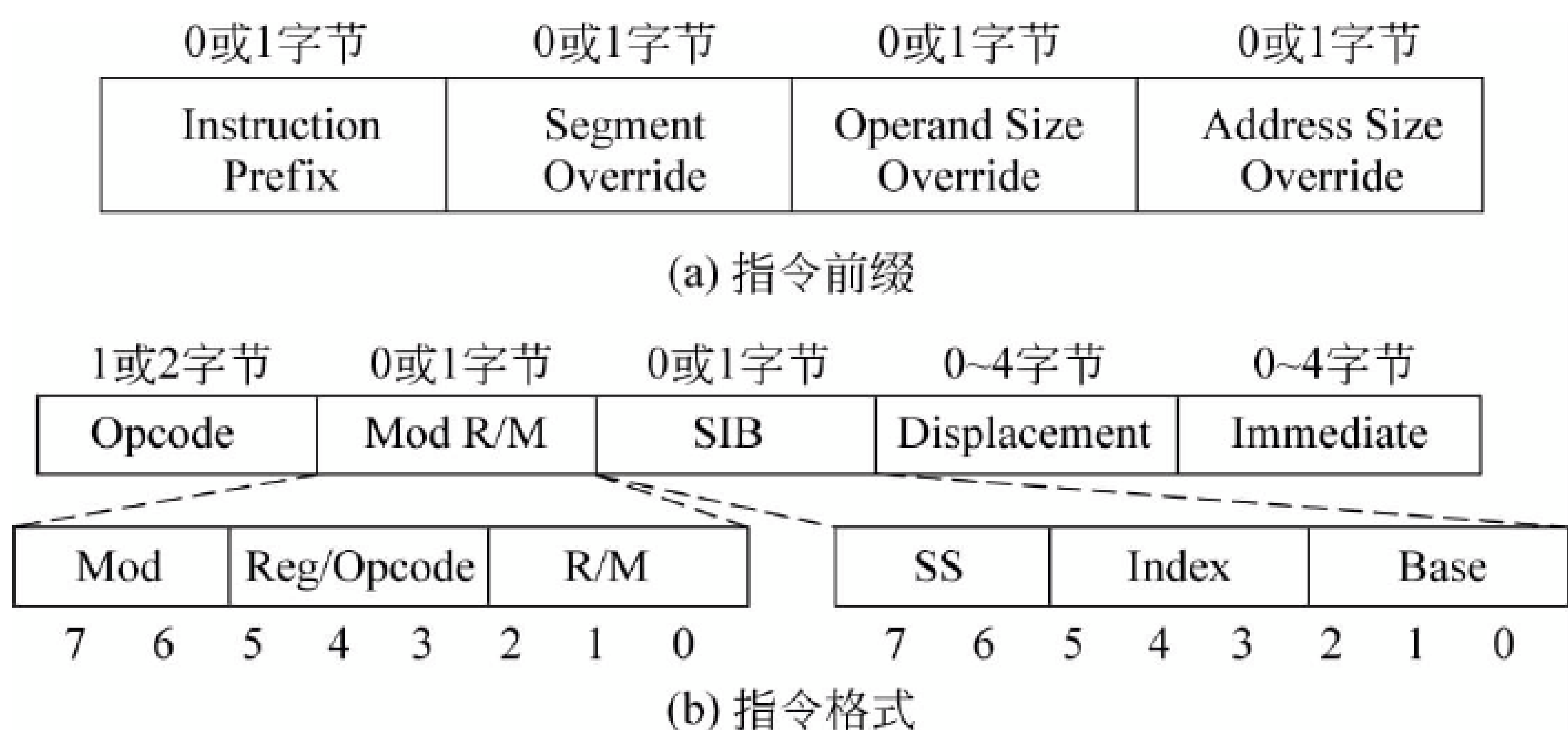


图 5.9 Pentium Ⅱ 的指令格式

(1) 如果 Pentium 计算机的指令带有自己的前缀 Instruction Prefix,可由 1~4 个重复出现的 LOCK 部分组成,用于保证在多处理机环境下,处理机能以互斥的方式共享主存储器。通过这种办法处理字符串,比起使用软件办法实现要快得多。

(2) Segment Override 明确指出在指令中使用哪一个段寄存器。

(3) Operand Size Override 用于指出指令中的操作数是 16 位还是 32 位。

(4) Address Size Override 用于指出在本指令中计算存储器地址时使用 16 位还是 32 位的变址偏移量 Displacement。

下面给出指令格式各部分的功能简介。

(1) Pentium Ⅱ 的指令的操作码可以由 1~2 个字节组成,在指明操作功能的同时,有时还用于指出操作数的长度是字节、半字或者全字,存储器访问是读操作还是写操作,立即数是否需要符号位扩展等。

(2) ModR/M 字段和接下来的 SIB 字段用于给出寻址信息,例如数据是在寄存器中还是在存储器中,如果是在存储器中,本字节用于指出本指令使用的寻址方式,2 位的 Mod 和 3 位的 R/W 字段可以形成 32 个值,区分 8 个通用寄存器和 24 种变址方法;3 位的 Reg/Opcode 字段可以用于指定一个寄存器的编码,或者与 Mod 字段一起用于寻址方式的译码。

(3) SIB 字段可以与 Mod 字段一起用于寻址方式的译码,SIB 也被划分成 3 个子字段,2 位的 SS 用于指定变址寻址计算中的放大因子,3 位的 Index 和 Base 分别用于指定变址寄存器和基地址寄存器。

(4) Displacement 字段用于提供变址寻址方式下的偏移值,可以为 8、16 或 32 位长度。

(5) Immediate 字段用于提供立即数寻址方式下的立即数,可以为 8、16 或 32 位长度。

这套指令系统的特点:指令操作码还含有区分寻址方式的某些信息;变址寻址的 Offset 的字段可以为 1、2、4 个字节,可以在指令中支持访问更大的存储器空间。

下面分类列出完成整数运算和其他一些功能的常用指令,Pentium Ⅱ 计算机的指令系统混合了 32 位字长的指令和 16 位字长的指令模式,后者是为了与原有的 8086 机型保持软件兼容,其代价是使这套指令系统的实现更为复杂。

算术与逻辑运算指令	数据传送指令
ADD DST,SRT	MOVE DST,SRC
SUB DST,SRT	PUSH SRC
ADC DST,SRT	POP DST

SBB DST,SRT
AND DST,SRT
OR DST,SRT
XOR DST,SRT
TST SRC1,SRC2
CMP SRC1,SRC2
MUL SRC;无符号数除法
IMUL SRC;带符号数除法
DIV SRC;无符号数除法
IDIV SRC;带符号数除法
INC DST
DEC DST
NEG DST;变补码
NOT DST

控制转移指令
JMP ADDR
Jxx ADDR;条件转移
CALL ADDR
RET
IRET
LOOPxx;循环直到条件满足条件
INTxx
INTO

其他指令
SWAP DST
NOP;空操作
HALT;停机
IN AL,PORT;输入
OUT PORT,AL;输出
WAIT;等待中断

XCHG DST,SRC;内容交换
LEA DST,SRC;存 SRC 的优先地址到 DST
CMOV DST,SRC;比较 2 个串的内容

逻辑移位和循环移位指令
SAL DST, # ;算术左移
SAR DST, # ;算术右移
SHL DST, # ;逻辑左移
SHR DST, # ;逻辑右移
ROL DST, # ;循环左移
ROR DST, # ;循环右移
RCL DST, # ;带进位 C 左移
RCR DST, # ;带进位 C 游移

字符串指令
LODS
STOS
MOVS
CMPS
SCAS

码指令
STC
CLC
CMC
STD
CLD
STI
CLI
PUSHFD;状态寄存器内容进堆栈
POPFD;状态寄存器内容出堆栈

这里的符号含义说明：DST 是指目的操作数地址；SRC 是指源操作数地址；“#”是指移位的位数。

5.3.2 MIPS32 计算机的指令系统

MIPS(Microprocessor without Interlocked Pipeline Stages)是 20 世纪 80 年代中期推出的 RISC 结构的计算机系统，多年来形成了 32 位或 64 位字长、功能各异的几个型号的产品，我们选择其中的 MIPS32 为例进行介绍。

MIPS32 是 32 位字长、典型 RISC 结构指令系统的计算机。其指令格式简单，指令数量较少，只选用 3 种基本寻址方式，通用寄存器较多，编译系统简单高效，更方便实现指令流水。3 种指令格式如图 5.10 所示。

在 MIPS32 型号的系统中，还提供一套 16 位字长的 MIPS16e 指令系统，以便更方便地支持嵌入式系统的使用要求。

第 1 种指令称为 **R 型指令**，完成的功能例如 rd← rs op rt

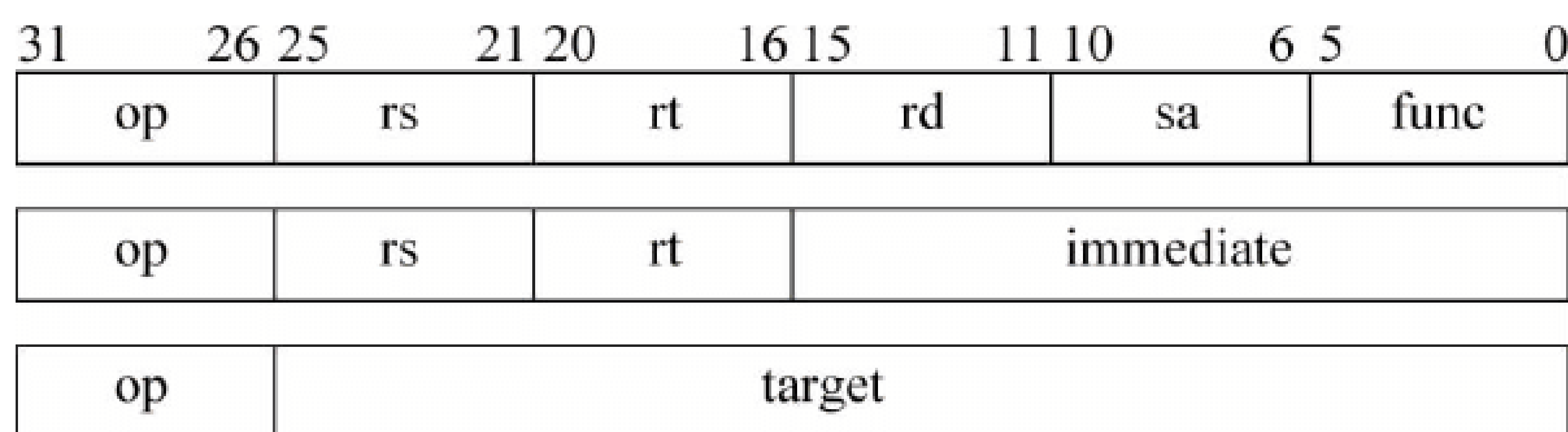


图 5.10　MIPS32 机的指令格式

rd←(rs<rt)

即 rs 的内容比 rt 的内容小则 rd 置 1,否则清 0。

第 2 种指令称为 **I 型指令**,完成的功能例如 rt← rs op immediate

rt←memory[base+offset]

memory[base+offset]←rt

若 rt=rs,转移到地址 PC+offset×4,否则顺序执行;

若 rt<>rs,转移到地址 PC+offset×4,否则顺序执行。

第 3 种指令称为 **J 型指令**,完成无条件跳转的操作功能。用 PC 的最高 4 位拼接 target ×4 的值作为跳转的指令地址。

下面给出某些典型指令的例子,只提供汇编语句格式,并不说明这些指令的具体功能。

```
add $1,$2,$3,     addi $1,$2,100,    addu $1,$2,$3,    addiu $1,$2,100
sub $1,$2,$3,     subu $1,$2,$3
mult $2,$3,       multu $2,$3,       div $2,$3,        divu $2,$3
mfhi $1,          mflo $1,           mfco $1,$epc,
and $1,$2,$3,     or $1,$2,$3,       andi $1,$2,100,   ori $1,$2,100
sll $1,$2,10,     srl $1,$2,10
lw $1,100($2),    sw $1,100($2),     lui $1,100
beq $1,$2,100,    bne $1,$2,100,
slt $1,$2,$3,     slti $1,$2,100,    sltu $1,$2,$3,    sltiu $1,$2,100
j 10000,          jr $31,            jal 10000
```

5.3.3　PDP-11 计算机的指令系统

PDP-11 是 20 世纪 60 年代初由美国 DEC 公司研制成功的小型(当时情况)系列机,曾被广泛应用。1978 年,DEC 公司在此基础上又推出 32 位字长、性能更高的 VAX-11 系列机。

PDP-11 计算机的主要特点体现在以下 5 个方面。

(1) 采用单总线(UNIBUS)结构,即中央处理机、主存储器、全部外围设备都接在唯一的一组总线上,优点是硬件结构简单,易于扩展,缺点是系统运行效率低。

(2) 该计算机的机器字长 16 位;CPU 内有 8 个通用寄存器,分别用 R_0～R_7 表示,但 R_6 和 R_7 分别为堆栈指针 SP 和程序计数器 PC,真正能通用的只有 R_0～R_5。

(3) 该机可以按字或字节寻址,指令中有处理字类型数据和处理字节类型数据的两类指令。

(4) 操作码使用逐段扩展技术,不同的指令使用不同位数(长度)的操作码,从 4 位到 16

位不等，在很短的指令字中充分发挥每一位的效用。

(5) 在指令的操作数地址部分，采用统一寻址方式访问通用寄存器、主存及外围设备，主存和外围设备统一编址，没有专门的I/O指令。指令中用6个二进制位(bit)表示一个操作数的地址，如图5.11所示。

Mod	R
3位	3位

图5.11 操作数的地址格式

其中，Mod用于确定寻址方式，R用于指定寄存器编号，各占3位，能分别表示8种不同的寻址方式和8个不同的寄存器。R的取值从000到111，分别表示R_0到R_7，其中R_6和R_7也可以写成SP(堆栈指针)和PC(程序计数器)，真正能通用的只有R_0～R_5。

Mod的8个取值分别代表8种寻址方式，具体如表5.2所示。

表5.2 Mod的取值与寻址方式的对应关系

Mod取值	寻址符	寻址方式
000	R	寄存器寻址，R的内容即为操作数
001	(R)	寄存器间接寻址，R的内容是操作数的地址
010	(R)+	寄存器间接寻址并自动增量，R的内容是操作数的地址，用后R的内容执行增1(对字节指令)或增2(对字指令)
011	@(R)+	间接寻址并自动增量，R的内容是操作数地址的地址，用后R的内容增1(对字节指令)或增2(对字指令)
100	-(R)	自动减量的寄存器间接寻址，修改后的R的内容是操作数的地址，即用前先对R的内容执行减1(对字节指令)或减2(对字指令)
101	@-(R)	自动减量并间接寻址，修改后的R的内容是操作数地址的地址，即用前先对R的内容执行减1(对字节指令)或减2(对字指令)
110	X(R)	变址寻址，将X与R的内容相加之和作为操作数的地址
111	@X(R)	变址并间接寻址，将X与R的内容相加之和作为操作数地址的地址

可以看到，在Mod的高两位取值相同、仅最低位的取值不同的两种寻址方式中，最低位取值为1的寻址方式，总是在前一种寻址方式上多了一次间接寻址操作。

还可以发现，不能把2、3、6、7这4种寻址方式按照上述规定用于R_7，即程序计数器PC，为此可以把27、37、67、77这几个组合变通一下，用于另外的寻址方式，如表5.3所示。

表5.3 另外几种寻址方式

Mod取值的组合	寻址符	寻址方式
27	#n	立即数寻址，#n为立即数，放在指令的第二个字中
37	@#A	直接地址寻址，#A为地址，放在指令的第二个字中
67	X	相对寻址，(当前指令地址+4)+X为有效地址，X在指令的第二个字中
77	@X	相对加间接寻址，(当前指令地址+4)+X为地址的地址，X在指令的第二个字中

有了上述介绍，下面就可以简明地对PDP-11计算机的指令格式进行分类说明。

(1) 双操作数指令，采用4位操作码，每个操作数形式地址占6位，如图5.12所示。

操作码		SS(源操作数)		DD(目的操作数)	
OP	Mod	R	Mod	R	
15 14 13 12	11 10 9	8 7 6	5 4 3	2 1 0	

01 SS DD MOV　　11 SS DD MOVB

06 SS DD ADD　　16 SS DD SUB

图 5.12　双操作数指令格式

多数情况下，操作码最高一位的值，用于区分是字指令还是字节指令，为 1 则是字节指令，并在汇编语句名后用跟字符 B 标记(下同)；加、减指令不支持字节运算，故可以用 06 和 16 操作码分别表示 ADD 和 SUB 指令。

(2) 单操作数指令，采用 10 位操作码，一个操作数形式地址占 6 位，如图 5.13 所示。

操作码	DD(目的操作数)	
OP	Mod	R

0050 DD CLR　　1050 DD CLRB

0001 DD JMP　　0030 DD SWAP

图 5.13　单操作数指令格式 1

另一组单操作数指令，采用 10 位操作码，一个操作数形式地址占 6 位。这里的操作数可能为目的操作数、源操作数、立即数，由指令本身决定，如图 5.14 所示。

操作码	DD/NN/SS	
OP	Mod	R

0060 DD ROR　　1060 DD RORB

0061 DD ROL　　1061 DD ROLB

图 5.14　单操作数指令格式 2

(3) 无操作数指令，指令字的全部 16 位都用作操作码，如图 5.15 所示。

操作码
OP

000000 HALT　　104000~104377 EMT

000001 WAIT　　104400~104477 TRAP

图 5.15　无操作数指令格式

(4) 特殊格式的指令，包括以下 3 种类型。

① 双操作数指令，采用 7 位操作码，其中的一个操作数形式地址只用 R 表示。

② 单操作数指令，采用 13 位操作码，操作数形式地址只用 R 表示。

③ 其他特殊指令，主要是相对转移指令，采用 8 位操作码和 8 位补码的偏移量表示。

5.3.4 教学计算机的指令系统

1. 确定教学计算机指令系统的原则

合理地确定一台计算机的指令系统，无论对计算机厂家还是对最终用户来说都是十分重要的事情。它密切关系到计算机设计与实现的复杂程度和生产成本，计算机使用的难易程度和运行效率。对主要用于教学实例和教学实验目的的 16 位字长的计算机，更多地应关

注它在教学过程中的作用和使用方法。设计时的主要思想如下。

(1) 尽可能精小的指令集,指令数目要适当地少,较短的指令格式,简单的寻址方式,单字指令为主,每条指令的功能要尽可能地简单。

(2) 指令系统要有一定的完备程度,有较好的典型性,给出的指令格式适当规范,指令分类合理,符合人们通常的编程使用习惯,指令执行步骤容易理解。

(3) 更高的可扩充性,即为学生添加各种新的指令留下比较充足的余地,包括为每一类指令保留多条要学生亲自实现的指令。

(4) 符合课程教学用计算机的特定要求。选用16位字长指令格式,指令操作码固定为8位长度,实现这套指令系统时,计算机硬件组成和指令执行流程要简单。

2. 教学计算机的指令系统说明与指令分类

1) 指令格式

教学计算机的指令系统有单字和双字指令,第一个指令字的高8位是指令操作码,低8位和双字指令的第二个指令字是操作数地址字段,分别有3种用法,如图5.16所示。



图5.16 教学计算机的指令格式

8位的指令操作码(记作“IR_{15}～IR_8”)中各位的含义如下。

(1) IR_{15}、IR_{14}用于区分指令组:0X表示A组,10表示B组,11表示D组。

(2) IR_{13}用于区分基本指令和扩展指令:基本指令该位为0,扩展指令该位为1。

(3) IR_{12}用于简化控制器实现,暂定该位的值为0。

(4) IR_{11}～IR_8用于区分同一指令组中的不同指令。

操作数地址字段的设计结果如下。

(1) DR(目的寄存器)和SR(原寄存器)是4位的寄存器编号,可寻址16个寄存器。

(2) 8位的IO端口地址,默认的串行接口地址为十六进制的80和81。

(3) 8位的相对寻址偏移量offset,转移范围从−128～+127。

(4) 16位的立即数由指令的第2个字提供,用于把这个数值传送到选定的寄存器中。

(5) 16位的直接地址由指令的第2个字提供,作为指令转移地址或子程序入口地址。

(6) 16位的变址偏移量由指令的第2个字提供,用于以变址寻址方式访问内存。

2) 指令分类

按不同的分类标准,可以把16位机的指令划分成不同的指令组,具体如下。

(1) 从指令字长度区分,有单字指令和双字指令,也允许定义与使用3字指令。

(2) 从操作数的个数区分,有双操作数指令、单操作数指令和无操作数指令。

(3) 从使用的寻址方式区分,有寄存器寻址、寄存器间接寻址、立即数寻址、直接寻址、变址寻址、相对寻址、堆栈寻址等多种基本寻址方式的不同类别指令。

(4) 从指令执行步骤区分,有采用2步、3步、4步完成的这样3组。

(5) 从指令功能区分,如表5.4所示。

表 5.4 指令从功能划分所包含的类型

指令类型	包含的指令
算术和逻辑运算类指令	ADD、SUB、AND、OR、XOR、CMP、TEST、DEC、INC、ADC、SBB、NOT
数据传送类指令	MVRR、MVRD
移位类指令	SHL、SHR、RCL、RCR、ASR
读写内存类指令	LDRR、STRR、PUSH、POP、PSHF、POPF、LDPC、LDRA、STRA、LDRX、STRX
输入/输出类指令	IN、OUT
指令流程控制类指令	JR、JRC、JRNC、JRZ、JRNZ、JMPA、CALA、RET、JRS、JRNS、JMPR、CALR、IRET
其他指令	CLC、STC、EI、CI

教学计算机系统实现了其中的 30 条基本指令,用于支持监控程序和简单的汇编语言程序设计。保留了其余 19 条扩展指令供学生在教学实验中进行扩展,即完成对这些指令的设计与调试,当然,还可以扩展另外一些指令。

在表 5.5 和表 5.6 中分别汇总了前面介绍的基本指令(已经实现,属于基本配置)和扩展指令(已经实现,属于可选配置)的有关内容。表中的分组是按照指令的执行步骤来划分的,A 组指令用 2 步完成,B 组指令用 3 步完成,D 组指令用 4 步完成。

表 5.5 基本指令汇总表

指令格式	汇编语句	操作数个数	CZVS	分组	功能说明
00000000 DRSR	ADD DR,SR	2	****	A组指令	DR←DR+SR
00000001 DRSR	SUB DR,SR	2	****		DR←DR−SR
00000010 DRSR	AND DR,SR	2	0*0*		DR←DR and SR
00000011 DRSR	CMP DR,SR	2	****		DR-SR
00000100 DRSR	XOR DR,SR	2	0*0*		DR←DR xor SR
00000101 DRSR	TEST DR,SR	2	0*0*		DR and SR
00000110 DRSR	OR DR,SR	2	0*0*		DR←DR or SR
00000111 DRSR	MVRR DR,SR	2	••••		DR←SR
00001000 DR0000	DEC DR	1	****		DR←DR−1
00001001 DR0000	INC DR	1	****		DR←DR+1
00001010 DR0000	SHL DR	1	*•••		DR,C←DR×2
00001011 DR0000	SHR DR	1	*•••		DR,C←DR/2
01000001 OFFSET	JR ADR	1	••••		无条件跳转到 ADR
01000100 OFFSET	JRC ADR	1	••••		C=1 时跳转到 ADR
01000101 OFFSET	JRNC ADR	1	••••		C=0 时跳转到 ADR
01000110 OFFSET	JRZ ADR	1	••••		Z=1 时跳转到 ADR
01000111 OFFSET	JRNZ ADR	1	••••		Z=0 时跳转到 ADR
10000000 0000000 ADR(16 位)	JMPA ADR	1	••••		无条件跳到 ADR
10001000 DR0000 DATA(16 位)	MVRD DR,DATA	2	••••		DR←DATA
10000010 I/O PORT	IN I/O PORT	1	••••		R0←[I/O PORT]
10000110 I/O PORT	OUT I/O PORT	1	••••		[I/O PORT]←R0

续表

指令格式	汇编语句	操作数个数	CZVS	分组	功能说明
10001001DRSR	LDPC DR,[SR]	2	••••	B组指令	PC←[SR]
10000001DRSR	LDRR DR,[SR]	2	••••		DR←[SR]
10000011DRSR	STRR [DR],SR	2	••••		[DR]←SR
10000100 00000000	PSHF	0	••••		FLAG 入栈
10000101 0000SR	PUSH SR	1	••••		SR 入栈
10000111 DR0000	POP DR	1	••••		DR←出栈
10001100 00000000	POPF	0	****		FLAG←出栈
10001111 00000000	RET	0	••••		子程序返回
11001110 00000000 ADR(16 位)	CALA ADR	1	••••	D组指令	调用首地址为 ADR 的子程序

注：① 表中 CZVS 一列，“*”影响状态位，“•”不影响状态位。

② 运算器中有 16 个累加器 R_0～R_{15}，其中 R_4 用作堆栈指针 SP；IN、OUT 指令默认使用 R_0，对其他指令 R_0 是通用寄存器；其余寄存器用作通用寄存器，即指令中的 DR、SR。

③ LDPC 是特权指令，只用在监控程序中，不能在用户程序中使用。

表 5.6　扩展指令汇总表

指令格式	汇编语句	操作数个数	CZVS	分组	功能说明
00100000 DRSR	ADC DR,SR	2	****	A组指令	DR←DR+SR+C
00100001 DRSR	SBB DR,SR	2	****		DR←DR−SR−C
00101010 DR0000	RCL DR	1	*•••		DR 带进位 C 循环左移
00101011 DR0000	RCR DR	1	*•••		DR 带进位 C 循环右移
00101100 DR0000	ASR DR	1	*•••		DR←DR 算术右移
00101101 DR0000	NOT DR	1	0*0•		DR←$\overline{DR}$
01100000 0000SR	JMPR SR	1	••••		跳转到 SR 给的地址
01100100 OFFSET	JRS ADR	1	••••		S=1 时跳转到 ADR
01100101 OFFSET	JRNS ADR	1	••••		S=0 时跳转到 ADR
01101100 00000000	CLC	0	0•••		C=0
01101101 00000000	STC	0	1•••		C=1
01101110 00000000	EI	0	••••		开中断，INTE←1
01101111 00000000	DI	0	••••		关中断，INTE←0
11100100 DR0000 ADR(16 位)	LDRA DR,[ADR]	2	••••	B组指令	DR←[ADR]
11100111 0000SR ADR(16 位)	STRA [ADR],SR	2	••••		[ADR]←SR
11100101 DRSR ADR(16 位)	LDRX DR,OFFSET[SR]	2	••••	D组指令	DR←[DATA+SR]
11100110 DRSR ADR(16 位)	STRX DR,OFFSET[SR]	2	••••		[DATA+SR]←SR
11100000 0000SR	CALR SR	1	••••		调用 SR 指明的子程序
11101111 00000000	IRET	0	••••		中断返回

注：① 表中 CZVS 一列，“*”表示对应的状态位在该指令执行后会被重置；“•”状态位在该指令执行后不会被修改。

② 扩展指令的功能、格式、操作码和操作数地址字段的确定，留给同学自己设计。表中给出的只是可能的一种选择，但同学们一定要认识到，这里的基本指令和扩展指令合在一起共同组成教学计算机完整的指令系统，彼此需要协调，至少彼此之间不能有冲突。

5.4 教学计算机的汇编语言程序设计

5.4.1 汇编语言及其程序设计中的有关概念

汇编语言大体上是对计算机机器语言(二进制指令代码)进行符号化的一种表示方式,每一个基本汇编语句对应一条机器指令,在此基础上,再增加一些扩展功能,诸如支持系统调用,允许定义和使用宏(MACRO),可以定义和使用伪指令等,以便能更方便地完成汇编语言程序设计。在进行汇编语言程序设计时,可以直接使用英文单词或其缩写表示指令,使用标识符表示数据或者地址,有效地避免了记忆二进制的指令代码,不再由程序设计人员为指令和数据分配内存地址,直接调用操作系统的某些程序段完成输入输出、建立与读写文件等操作功能。用编辑程序建立好的汇编语言源程序,需要经过系统软件中的汇编程序"翻译"为机器语言程序之后,才能交付给计算机硬件系统去执行。

要完成汇编语言程序设计,需要对程序结构中的流程控制、实现这些流程经常使用的语句有个初步的了解。与用其他语言设计的程序一样,在汇编语言的程序中,也包括顺序执行、无条件转移执行、条件分支执行、循环执行、子程序调用与返回执行5类程序结构,如图5.17所示。对这5种程序结构简介如下。

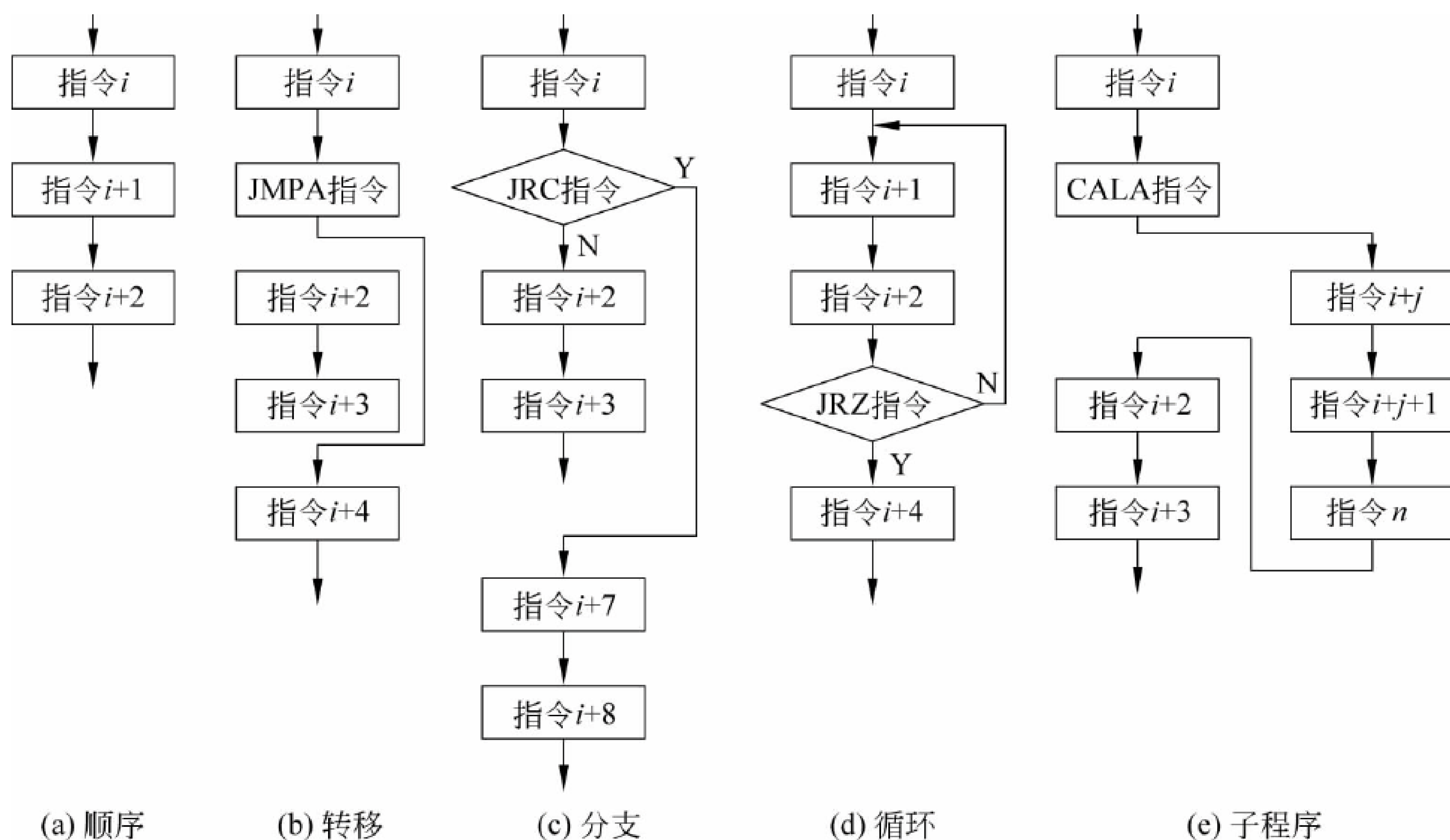


图5.17　5种常用到的程序执行流程表示

(1) 顺序执行。按照汇编语句(机器指令)在程序中排列的顺序从前向后逐条执行,为此只需把顺序执行的指令在程序中按次序编写即可。顺序执行的指令所占比例最高。

(2) 无条件转移执行。表示需要改变指令的执行次序,下一条将不执行排在该指令之后的那条指令,而是用指令中给出的转移地址去找到相应指令并执行,如JMPA和JR指令。

(3) 条件分支执行。表示需要按照指令中指出的条件为真(成立)还是为假(不成立),

从两条指令中选择其一来执行,条件成立则转移,不成立则顺序执行,如JRC、JRNC、JRZ、JRNZ指令。

(4)循环执行。是让一段程序反复多次地重复执行的控制流程,直到达到需要重复执行的次数,或者某一个条件出现时结束重复过程。例如,在实现按次数控制循环过程时,可以在一个寄存器中赋一个正的初值作为循环次数,每循环一次使该寄存器的内容减1,直到该寄存器的值变为0则结束循环。另一个典型例子,在使用串行接口时,通过读入接口状态寄存器的内容并判别某一位的值,以决定开始执行数据输入输出操作,还是继续执行读入接口状态并检查的循环过程,控制循环执行可能用到无条件转移指令或者条件转移指令。

(5)子程序调用与返回执行。子程序是通过子程序调用语句使其投入运行过程的一个特殊的程度段。教学计算机中的CALA指令就是子程序调用语句,它的作用是暂停主程序的执行过程,转去执行一个子程序,待子程序执行之后,还要返回来接着执行停下来的主程序。在子程序中要用RET指令表明子程序结束,以便启动返回操作。

例如,对分支结构,用JRC指令控制时,如果标志位C=1则分支到指令$i+7$,否则执行下邻指令$i+2$。

对循环结构,例如用JRZ指令控制时,指令$i+1$、$i+2$和JRZ指令构成循环体,JRZ指令用于控制继续循环还是结束循环。

对子程序结构,例如在主程序中用CALA指令调用由指令$i+j$到指令n(指令n一定是RET指令)组成的子程序,当子程序结束执行过程后,程序必将返回到紧跟在CALA指令之后的指令$i+2$处,再次开始主程序的执行过程。

5.4.2 教学计算机的汇编程序设计举例

1. 指令功能与在程序中的作用

这里主要想表明每个语句(每条指令)所完成的功能是什么,在汇编程序中怎样组合语句实现一定的运算处理功能这样两个方面的问题。这对从没有接触过汇编语言程序设计的同学有启蒙作用。大部分指令(至少是每一类指令)都出现在下面的例子中。

【例5.1】 本程序的功能是在显示器屏幕上循环显示95个可打印字符(包括空格字符)。

```
A 2000                      ;从内存的 2000 单元开始建立用户的第一个程序
2000:  MVRD   R1,7E         ;向寄存器传送直接数
2002:  MVRD   R0,20
2004:  OUT    80            ;通过串行接口输出 R0 低位字节内容到显示器屏幕
2005:  PUSH   R0            ;保存 R0 寄存器的内容到堆栈中
2006:  IN     81            ;读串行接口的状态寄存器的内容
2007:  SHR    R0            ;R0 寄存器的内容右移一位,最低位的值移入标志位 C
2008:  JRNC   2006          ;条件转移指令,当标志位 C 不是 1 时就转移到 2006 地址
2009:  POP    R0            ;从堆栈中恢复 R0 寄存器的原内容
200A:  CMP    R0,R1         ;比较两个寄存器的内容是否相同,相同则标志位 Z=1
200B:  JRZ    2000          ;条件转移指令,当标志位 Z 为 1 时就转移到 2000 地址
200C:  INC    R0            ;把 R0 寄存器的内容增加 1
200E:  JR     2004          ;无条件转移指令,一定转移到 2004 地址
200F:  RET                  ;子程序返回指令,教学计算机的用户程序也必须用 RET 指令结束
```

【例 5.2】 本程序的功能是首先把字符 A～F 写到内存的 2040～2045 几个单元,之后再读出来并显示到屏幕上。

```
A 2020
2020:  MVRD  R3,06          ;给出写内存操作的次数
2022:  MVRD  R2,203F
2024:  MVRD  R1, 40
2026:  INC   R2             ;给出写内存操作的内存地址
2027:  INC   R1             ;给出写内存操作的数据内容
2028:  STRR  [R2],R1        ;写寄存器 R1 的内容到由 R2 指定地址的内存单元中
2029:  LDRR  R0,[R2]        ;读出内存单元的数据到 R0 寄存器
202A:  OUT   80
202B:  IN    81
202C:  SHR   R0
202D:JRNC    202B
202E:  DEC   R3             ;检查 6 次写内存操作是否完成
202F:  JRNZ  2026           ;未完则开始下一次写内存操作
2030:  RET                  ;程序结束
```

【例 5.3】 本程序的功能是从键盘输入字符并送到显示器屏幕显示,说明如何使用子程序调用指令和转移指令。在子程序中,可以完成变大写英文字母为小写字母并将其显示出来的功能。

```
A 2040                      ;在该程序中给出了调用子程序和设计子程序的例子
2040:  IN    81             ;此处的 4 个语句检查有无敲击过键盘上的一个键
2041:  SHR   R0
2042:  SHR   R0
2043:  JRNC  2040
2044:  IN    80             ;把从键盘输入的一个字符输入到 R0 低位字节
2045:  OUT   80             ;输出 R0 低位字节内容到显示器屏幕
2046:  PUSH  R0
2047:  IN    81
2048:  SHR   R0
2049:  JRNC  2047
204A:  POP   R0
204B:  CALA  2050           ;调用子程序, 子程序的入口地址为 2050
204D:  JMPA  2040           ;转移指令, 转移地址为 2040
204A:  RET
                            ;以下是一个子程序
2050:MVRD    R1,20
2051:  ADD   R0,R1          ;修改输入的字符的编码,可变大写字母为小写字母
2052:  OUT   80             ;把修改过的字符输出到显示器屏幕
2053:  RET                  ;子程序返回指令,每个子程序都应该用 RET 结束
```

上面 3 个程序使用不同的内存区域,是为了确保它们可以同时存储在内存中,以便可以随时选择其一来运行,彼此之间不会有冲突。

2. 汇编语言程序设计中的有关概念

1）教学计算机的机器语言

教学计算机目前提供了约50条基本机器指令，其中约30条的指令格式、功能已完全确定，并已用组合逻辑与微程序两种控制方案实现，可供用户编程使用；尚保留另外约20条的指令交由实验者自己去定义与实现。

从计算机组成原理课的教学与实验的角度考虑，不会要求学生编写较长的程序。对一些实验，一般十几行到几十行基本够用，此时，是可以用机器语言直接编程的。当教学计算机系统中没有接入计算机终端（或PC仿真终端）等输入输出设备时，实验者只能用开关向计算机内拨入由机器指令组成的程序。这种方式下的操作与实验效率是很低的，我们提供但并不推荐这种运行方式。

要用机器语言写程序，必须非常清楚所用机器指令的格式，具体操作码的编码，操作数形式地址的编码表示和每条指令的功能。我们已在5.3.4节以表格形式给出这些指令的机器码。

2）教学计算机的汇编语言

如何使用16位机系统的汇编语言，与系统的硬件配置有关。

当系统中不包括主要作为仿真终端使用的PC，而是使用计算机终端设备时，就只能使用监控程序提供的单条汇编命令A，通过计算机终端，逐条输入并汇编每一个汇编语句，使用起来还算是方便，缺点是不能支持伪指令，即不能使用符号代表变量与汇编语句标号，一般情况下要使用直接地址，对设计较小的汇编程序还是可以忍受的，对较长一点的汇编程序则要困难一些。具体操作方法如下。

（1）当教学计算机系统加电并正常启动起来之后，从键盘上输入监控程序的A（ASSEMBLY）命令，格式为：A跟一个空格字符再跟指定的内存地址并回车。

（2）接下来，实验者可逐条输入所要的汇编语句。这种用法与PC上的DEBUG很类似，系统会逐语句进行汇编，产生教学计算机的指令代码，并从给定地址逐条连续存放。若发现给出的汇编语句有错，则指出错误位置，并要求重新输入正确的语句。这个过程一直持续到实验者在应该输入语句的位置不再输入语句而直接按回车键结束。

请看下面给出的教学计算机的几个汇编语言程序的例子。

【例5.4】 设计一个程序，在屏幕上输出显示一个字符6。

```
A 2000                        ;地址从十六进制的 2000(内存 RAM区的起始地址)开始
2000: MVRD    R0,0036         ;把字符 6的 ASCII码送入 R0
2002: OUT     80              ;在屏幕上输出显示字符 6,80为串行接口地址
2003: RET                     ;每个用户程序都必须用 RET指令结束
2004:                         (按回车键即结束源程序的输入过程)
```

这就建立了一个从主存2000h地址开始的小程序。在这种方式下，所有的数字都约定使用十六进制数，故数字后不用跟字符h。每个用户程序的最后一个语句一定为RET汇编语句，因为监控程序是选用类似子程序调用方式使实验者的程序投入运行的，用户程序只有用RET语句结束，才能保证用户程序运行结束时能正确返回到监控程序的断点，保证监控程序能继续控制实验计算机的运行过程。

每一行中第一列的4位十六进制的数字，是内存地址，其后面那条汇编语句汇编出来的

指令代码就保存在由这个地址选中的存储单元中;当中的部分是汇编语句;分号之后给出的是注释信息,用于提高程序的可读性,对程序的功能没有影响。

【例 5.5】 设计一个程序,用次数控制在终端屏幕上输出 0～9 这 10 个数字符。

```
A 2020
       MVRD    R2,000A          ;送入输出字符的个数
       MVRD    R0,0030          ;0 字符的 ASCII 码
       OUT     80               ;输出保存在 R0 低位字节的字符
       DEC     R2               ;输出字符个数减 1
       JRZ     202E             ;判别 10 个字符输出完否,已完,则转移到程序结束处
       PUSH    R0               ;未完,保存 R0 的值到堆栈中
(2028) IN      81               ;查询串行接口状态,判字符的串行输出过程结束否
       SHR     R0;
       JRNC    2028             ;未完成,则循环等待
       POP     R0               ;已完成,准备继续输出下一字符,从堆栈恢复 R0 的值
       INC     R0               ;得到下一个要输出的字符
       JR      2024             ;转去输出字符
(202E)RET
```

这个程序只使用基本汇编语句。理解中的一个难点,是程序当中判别串行口是否完成一个字符的输出过程并循环等待的 3 个汇编语句。具体解释见教材中有关串行接口的内容。

该程序的执行码保存在 2020 起始的内存区中。若送入源码的过程中有错,系统会进行提示,等待重新输入正确语句。输入过程中,在应输入语句的位置直接按回车则结束输入过程。

接下来可用 G 2020 命令运行该程序。

思考题:当把 IN 81,SHR R0,JRNC 2028 这 3 个语句换成 3 个 MVRR R0,R0 语句,该程序执行过程会出现什么现象? 试分析并实际执行一次。

类似地,若要求在终端屏幕上输出 A～Z 共 26 个英文字母,应如何修改本例中给出的程序? 请验证。

【例 5.6】 从键盘上连续输入多个属于 0～9 的数字符并在屏幕上显示,遇非数字符结束程序。

从地址 2040 开始输入下列程序。

```
A 2040
         MVRD    R2,0030      ;用于判别数字符的下界值
         MVRD    R3,0039      ;用于判别数字符的上界值
(2044)   IN      81           ;判别键盘上是否按了一个键
         SHR     R0           ;即串行口是否有了输入的字符
         SHR     R0
         JRNC    2044         ;尚没有输入则循环测试
         IN      80           ;把输入字符读到 R0 低位字节
         MVRD    R1,00FF
         AND     R0,R1        ;将 R0 的高位字节清 0
```

```
        CMP   R0, R2          ;判别输入的字符是否<字符 0
        JRNC  2053            ;是,则转到程序结束处
        CMP   R3, R0          ;判别输入的字符是否>字符 9
        JRNC  2053            ;是,则转到程序结束处
        OUT   80              ;输出刚输入的数字符
        JMPA  2044            ;转去程序前边 2044 处等待输入下一个字符
(2053)RET
```

思考题,本程序中为什么不必判别串行口输出完成否?设计读入 A～Z 和 0～9 的程序,遇其他字符结束输入过程。

【例 5.7】 计算 1～10 的累加和。

```
A 2060
        MVRD  R1,0000         ;置累加和的初值为 0
        MVRD  R2,000A         ;最大的加数
        SUB   R3,R3           ;预置参加累加的数为 0
(2064)  INC   R3              ;得到下一个参加累加的数
        ADD   R1, R3          ;累加计算
        CMP   R3, R2          ;判别是否累加完
        JRNZ  2065            ;未完,开始下一轮累加
        RET
```

运行过后,可以用 R 命令看 R1 中的累加结果。

【例 5.8】 设计一个有读写内存和子程序调用指令的程序,功能是读出指定内存中的大写字母字符,将其显示到屏幕上,转换为小写字母后再写回存储器的原存储单元。

```
E20F0                         ;送入将被显示的 6个字符 A~F到内存 20F0开始的存储区域中
                              41 42 43 44 45 46
A 2080
        MVRD  R3,    0006     ;指定被读数据的个数
        MVRD  R2,    20F0     ;指定被读、写数据内存区首地址
(2084)  LDRR  R0,    [R2]     ;读内存中的一个字符到 R0 寄存器
        CALA  2100            ;调用子程序,入口地址为 2100,完成显示、字符转换和写回内存的
                              功能
        DEC   R3              ;检查输出的字符个数
        JRZ   208C            ;完成输出则结束程序的执行过程
        INC   R2              ;未完成,修改内存地址
        JR    2084            ;转移到程序的 2084 处,循环执行规定的处理
        RET

A 2100                        ;输入用到的子程序到内存 2100 开始的存储区
        OUT   80              ;输出保存在 R0 寄存器中的字符
        MVRD  R1,    0020     ;转换保存在 R0 中的大写字母为小写字母
        ADD   R0,    R1
        STRR  [R2],  R0       ;写 R0 中的字符到内存,地址同 LDRR 所用的地址
(2105)  IN    81              ;测试串行接口是否完成输出过程
```

```
SHR   R0
JRNC  2105              ;未完成输出过程则循环测试
RET                     ;结束子程序执行过程,返回主程序
```


运行过程中,可以直接看到屏幕上显示的内容,运行过后,再用 D 20F0 命令看内存的 20F0 区域中保存的运行结果,6 个大写的英文字母已经被修改为小写字母:

```
0061 0062 0063 0064 0065 0066
```

例 5.4～例 5.8 中,都是用监控程序的 A 命令完成输入源汇编程序的。在涉及汇编语句标号的地方,不能用符号表示,只能在指令中使用直接地址。使用内存中的数据,也由程序员给出数据在内存中的直接地址。显而易见,对这样的短小程序矛盾并不突出,当设计较长的程序时,一定会遇到相当大的困难。

在用 A 命令输入汇编源语句的过程中,有一定使用经验的人,常常抱怨 A 命令中未提供适当的编辑功能,这并不是设计者的疏漏。因为我们并不准备在这种操作方式下支持设计较长的程序,例如输入上述一些小程序,用监控程序的 A 命令完成,往往比用交叉汇编完成更方便。相反的情况是,若真的要设计较长的程序,就需要转到提供了交叉汇编程序的 PC 上去完成。例如,教学计算机的监控程序,就是通过 PC 的编辑程序建立源代码,用运行在 PC 中的交叉汇编程序完成汇编操作,从而得到由教学计算机的指令构成的执行程序的代码。在把得到的监控程序的执行码编程到教学计算机的 ROM 存储芯片之后,教学计算机就有了自己的监控程序。

3) 教学计算机的交叉汇编程序

交叉汇编的概念是用运行在另外一台计算机(例如 PC)中的程序,完成对本台计算机(例如教学计算机)的汇编语言的源程序执行汇编操作,并产生本台计算机(例如教学计算机)的执行程序的指令代码。显而易见,这是一种借鸡孵蛋的策略。采取这种办法有 2 个理由。一是基于教学计算机的可用资源非常有限,相比之下,作为教学计算机仿真终端使用的 PC 的资源和功能实在是太强大了,何不借用一下呢? 二是在教学计算机系统可以正常运行之前,新设计的、未经考验的硬件系统,与未完全设计好、又急于用于调试整机系统的监控程序撞到了一起,两个方面反反复复地修改是难免的,此时真的很难直接在教学计算机上实现一个自己的汇编程序,只能借助 PC 帮忙了。因此也可以说,在 PC 上实现用于教学计算机的交叉汇编程序,是研制开发教学计算机的一个过程性产品,原意是用于开发教学计算机的监控程序,后来作为已有成果继续使用下来。事实上,在教学计算机上,用该计算机的指令系统,开发一个汇编程序是一项比较容易的工作,监控程序可以为此提供足够的子程序,硬件系统也可以支持充足的内存空间,只是尚无暇顾及此事。在交叉汇编程序中,提供了汇编程序中那些最常用的基本功能,提供了最常用的伪指令,例如,说明变量、常量,分析与处理简单的表达式,保留存储区域,允许指明程序的起始地址和结束地址,允许使用语句标号等,这就为设计较长的汇编语言的程序提供了必要的支持。换言之,可以用比较正规的办法来设计教学计算机的汇编语言程序,最典型的实例就是教学计算机监控程序的源程序代码。交叉汇编程序有用 PC 汇编语言实现、用 C 语言实现和用 PASCAL 语言实现的 3 个不同的版本,可读性都比较好,扩展与修改功能更方便容易,是学习和理解汇编程序本身的设计与实现的良好技术资料。

4）教学计算机的高级语言和浮点数运算

在此之外，我们又在教学计算机上，用教学计算机的指令系统设计了BASIC语言的解释执行程序，从而实现了用软件方法支持的浮点数据运算，实现了真正意义上的表达式分析，实现了函数调用等功能，使主要用于教学实验的计算机有了运行高级语言程序的能力。这个BASIC语言的解释执行程序中含有比较多的知识量（这里的一些知识超出了本教材的教学要求），其源程序代码是值得读一读的，当然这需要花费一定的时间和精力。对有兴趣的师生，特别应该关注实现浮点数据运算的部分。事实上，可以比较方便地通过调用BASIC解释程序中的子程序，在教学计算机原有指令系统中加进浮点数运算的指令，进一步提高教学计算机的运算能力，这也是更深入地学习与理解浮点数运算规则、执行步骤的有效途径。

本章内容小结和学习方法建议

本章介绍设计计算机指令系统的基本标准，强调了一套好的指令系统对计算机厂家和用户都是至关重要的。本章以教学计算机指令系统作为重点例子，详细地讲解了指令的功能安排、指令格式、寻址方式选择3方面的知识。作为指令系统的实际例子，简单地列出了Pentium Ⅱ计算机、MIPS计算机和PDP-11计算机的基本指令的构成概貌，鼓励读者对比这几个指令系统在指令格式、寻址方式、指令构成等方面的特点和差异。

作为本课程的教学要求，汇编语言程序设计应该占有一定的分量，这更需要结合实际的指令系统来进行，教学计算机为此提供了必要的支持，把学习计算机组成与汇编语言程序设计两项内容有效地结合起来是值得提倡的。

比较好地熟悉了一套指令系统，再将其在自己设计的CPU系统中实现出来，能够运行自己设计的汇编程序，将学习到更多实用的知识和技术，可以体验学习的乐趣，享受成功后的喜悦，醉人的成就感将成为推动人们勇往直前的强力发动机。需要指出，规划、设计指令系统通常很难由计算机硬件人员完成，而把这套指令系统的功能在硬件系统上合理、高效地实现出来则是硬件人员的重要任务。

习题与思考题

1. 计算机硬件能直接识别和运行的只能是________程序。

   A. 机器语言　　B. 汇编语言　　C. 高级语言　　D. VHDL

2. 机器语言是面向________的，用它写的程序________在不同计算机之间移植。

   A. 计算机硬件　　B. 软件算法　　C. 容易　　D. 难以

3. 指令中用到的数据可以来自________。（可多选）

   A. 通用寄存器　　B. 微程序存储器　　C. 输入输出接口　　D. 指令寄存器

   E. 内存单元　　F. 磁盘

4. 汇编语言要经过________的翻译才能在计算机中执行。

   A. 编译程序　　B. 数据库管理程序　　C. 汇编器程序　　D. 文字处理程序

5. 在设计指令操作码时要做到________。（可多选）

A. 能区别一套指令系统中的所有指令　　B. 能表明操作数的地址

C. 长度随意确定　　D. 长度适当规范统一

6. 判断题,指出下面各题的对错,并简单说明理由。

(1) 寄存器寻址在指令中所占位数比较少,特别常用。

(2) 直接寻址在指令中所占位数比较少,特别常用。

(3) 变址寻址需要在指令中提供一个寄存器编号和一个数值。

(4) 相对寻址需要在指令中提供一个寄存器编号和一个数值。

(5) 堆栈寻址不是很有用。

(6) 指令的字长度对计算机的性能有较大影响。

(7) 计算机的指令越多、功能越强越好。

7. 某机器字长16位,主存按字节编址,转移指令采用相对寻址,由两个字节组成,第一字节为操作码字段,第二字节为相对位移量字段。假定取指令时,每取一个字节PC自动加1。若某转移指令所在主存地址为2000H,相对位移量字段的内容为06H,则该转移指令成功转移后的目标地址是________。

A. 2006H　　B. 2007H　　C. 2008H　　D. 2009H

8. 一条指令通常由哪两个部分组成? 指令的操作码一般有哪几种组织方式? 各自应用在什么场合? 各自的优缺点是什么?

9. 如何在指令中表示操作数的地址? 通常使用哪些基本寻址方式?

10. 什么是形式地址? 简述对变址寻址、相对寻址、基地址寻址应在指令中给出什么信息? 如何得到相应的实际(有效)地址? 各自有什么样的主要用法?

11. 下列关于RISC的叙述中,错误的是________。

A. RISC普遍采用微程序控制器

B. RISC大多数指令在一个时钟周期内完成

C. RISC的内部通用寄存器数量相对CISC多

D. RISC的指令数、寻址方式和指令格式种类相对CISC少

12. 下列给出的指令系统特点中,有利于实现指令流水的是________。

Ⅰ. 指令格式规整且长度一致Ⅱ. 指令和数据按边界对齐存放

Ⅲ. 只有Load/Store指令才能对操作数进行存储访问

A. 仅Ⅰ、Ⅱ　　B. 仅Ⅱ、Ⅲ　　C. 仅Ⅰ、Ⅲ　　D. Ⅰ、Ⅱ、Ⅲ

13. 堆栈的主要作用是什么? 如何完成读写堆栈的操作?

14. 寄存器寻址和寄存器间接寻址的区别是什么?

15. 偏移寻址通过将某个寄存器的内容与一个形式地址相加而生成有效地址。下列寻址方式中,不属于偏移寻址方式的是________。

A. 间接寻址　　B. 基址寻址　　C. 相对寻址　　D. 变址寻址

16. 为读写输入/输出设备,通常有哪几种常用的方式用以指定被读写设备?

17. MIPS16e计算机的指令系统与RISC计算机的指令系统有哪些类同的方面?

18. 用教学计算机的指令系统,设计一个程序,实现从键盘读入无符号的整型数据,到计算机内转换成二进制数并保存在累加器R0中,要求有适当的检查各种操作错误的能力。

19. 用教学计算机的指令系统,设计一个程序,实现从键盘读入有符号的整型数据,到

计算机内转换成补码形式的二进制数并保存在累加器 R0 中，要求有适当的检查各种操作错误的能力，例如输入了非法字符、数值溢出等。

20. 用教学计算机的指令系统，设计一个程序，实现 2 个无符号的整数相乘运算的功能。

21. 说明 CISC 和 RISC 两种指令系统各自追求的目标、特点以及对计算机硬件系统构成方面的影响，人们在说到指令系统时，常提到的 80% 和 20% 是什么含义？

22. 建立有关指令、指令系统、RISC、CISC、指令格式、指令操作码、指令操作数地址、形式地址、物理(实际)地址的概念。

23. 理解寻址方式、寄存器编址、存储器单元编址、IO 端口地址、指令周期、执行步骤的基本概念。

24. 了解机器语言、汇编语言、汇编语言程序设计、高级语言、程序执行流程，指令顺序执行、转移执行、子程序调用与返回的含义。

第6章 控制器部件

控制器部件是计算机的5大功能部件之一。其作用是向整机每个部件(包括控制器部件本身)提供它们协同运行所需要的控制信号,以确保计算机硬件系统能连续、自动地执行每一条指令,也就是执行程序。为了学懂控制器部件,需要掌握计算机各功能部件组成和它们的连接方式,能够理解各类信息在不同时间里、在计算机各功能部件之间的保存和流动的时间空间关系,如何划分指令的执行步骤、怎样处理执行步骤之间的衔接次序、确定每一个步骤实现的功能是学习的重点内容。

控制器包括硬布线控制器和微程序控制器两类,本章重点讲授计算机控制器的功能、组成与实现。将以指令的执行过程(步骤)为主线索,把通用原理性知识与真实计算机控制器实例相结合,以硬布线方案的控制器为主,并简单介绍微程序控制器的基础知识。

6.1 控制器的功能与组成概述

计算机硬件系统通常由运算器部件、控制器部件、存储器系统、输入设备和输出设备这5大部分组成。作为5大功能部件之一的控制器的作用,是向整机系统的每个部件(包括控制器部件本身)提供它们协同运行所需要的控制信号。计算机的核心功能是提供连续执行指令的能力,而每一条指令往往又要分成几个执行步骤才得以完成。由此又可以说,计算机控制器的基本功能,是依据当前正在执行的指令和它所处的执行步骤,形成并提供出在这一时刻整机各部件要用到的控制信号,这些控制信号是由各个部件的组成和运行要求决定的。

执行一条指令,通常总是要经过读取指令、分析指令、执行指令所规定的处理功能这3个阶段才能完成,这是在控制器的控制下实现的。控制器还要保证计算机能按程序中设定的指令运行次序,自动地连续执行指令序列。为此,控制器部件必须由几个具有不同处理功能的子部件组成,图6.1给出了控制器的基本组成,并且表明它在整机中的地位。

从图6.1可以看到计算机硬件系统的5大功能部件及其连接关系。虚线框内的是控制器部件,承担的是指挥、控制整个硬件系统的各个部件协同运行的任务。另外的4个实线框是4个执行部件,各自执行特定的功能,运算器执行数据运算,内存储器存储正在使用的程序和相应数据,输入设备用于从主机系统的外部向主机内送入信息,输出设备用于输出主机内部信息。这5个功能部件通过数据总线、地址总线和控制总线连接在一起,主要连接关系简单说明如下。

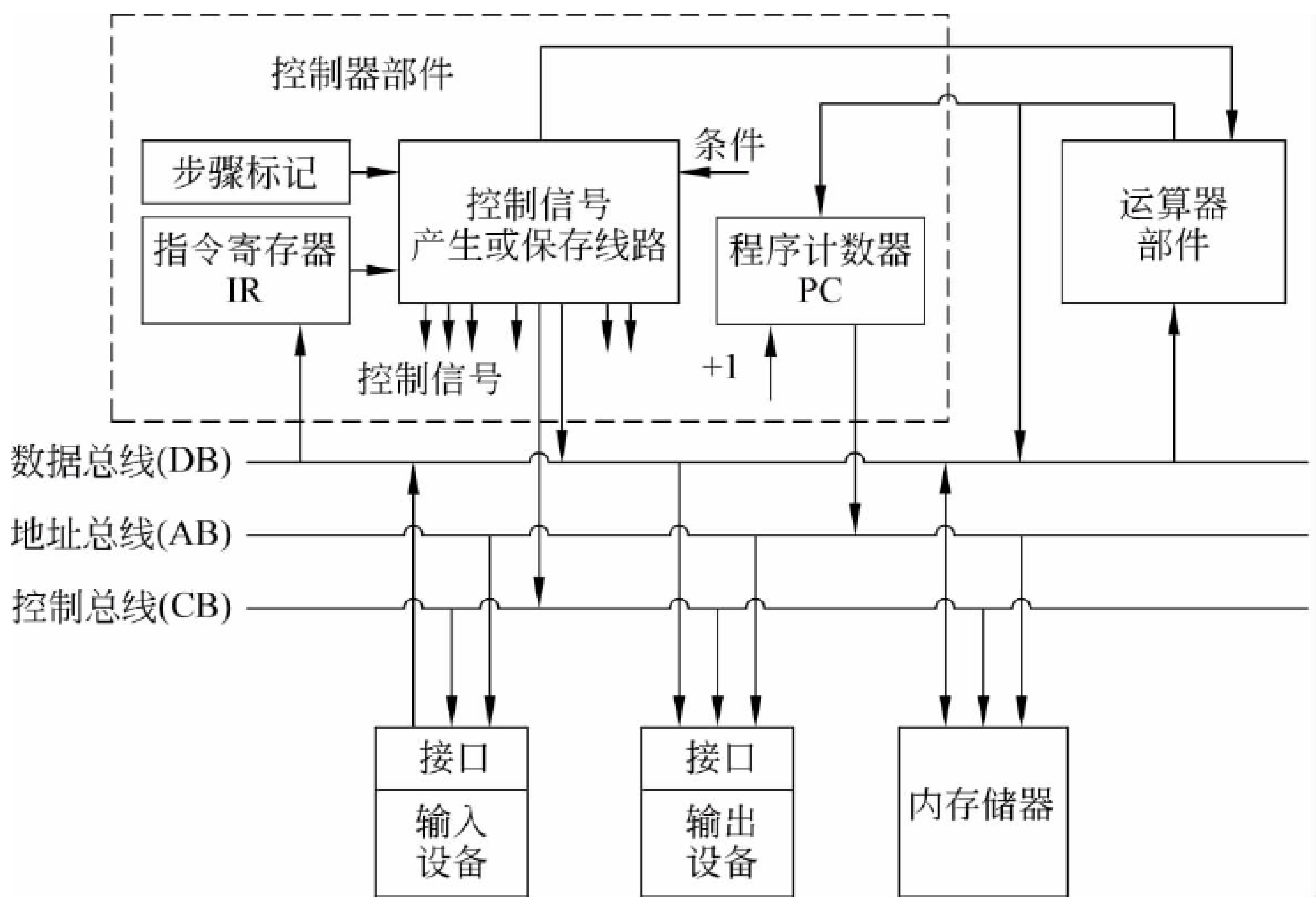


图 6.1　控制器组成与其在整机中的地位

(1) 运算器部件通过数据总线和内存储器、输入设备和输出设备传送数据。

(2) 输入和输出设备通过接口电路与总线相连接，进而接入主机系统。

(3) 内存储器、输入设备和输出设备从地址总线接收地址信息，从控制总线得到控制信号，通过数据总线与其他部件传送数据。

(4) 控制器部件从数据总线接收指令信息，从运算器部件接收指令转移地址，送出指令地址到地址总线。

(5) 控制器还要向系统中各部件提供它们运行所需要的控制信号，通常是通过控制总线向接口芯片、内存芯片提供控制信号。提供给运算器的控制信号是直接传送的。

图 6.1 中给出了组成控制器的 4 个基本子部件，它们分别如下。

(1) 程序计数器(PC)，用于提供指令在内存中的地址的部件，服务于读取指令，能执行内容增量和接收新的指令地址，解决的是指令执行的次序问题。

(2) 指令寄存器(IR)，用于接收并保存从内存储器读出来的指令内容的部件，在执行本条指令的整个过程中，为系统运行提供指令本身的主要信息，作为指令执行步骤和完成功能的基本依据。

(3) 指令执行的步骤标记线路，用于标记出每条指令的各个执行步骤的相对次序关系，解决的是每一条指令各步骤的转换和衔接关系问题。

(4) 全部控制信号的产生部件，它依据指令操作码、指令的执行步骤(时刻)，也许还有些另外的条件信号，来形成或提供出在指令的当前执行步骤下计算机各个部件要用到的控制信号。计算机整机各个执行部件正是在这些信号控制下协同运行，完成指令的执行功能，产生程序正确的执行结果。

在计算机正常执行指令的过程中，还要能够及时响应和处理一些随时可能出现的紧急事件，例如计算机运行遇到错误或故障、设备请求传送或接收数据、网上有信息送过来、由计算机实时控制的一个系统请求计算机立刻干预它的运行状态等，这些通常是以硬件或软件中断方式表现出来并进行处理的，这些内容不在这里讲解，将其放到第 11 章再详细说明，以

便首先把主要精力聚焦到指令本身的执行过程。

依据前述控制器最后两个组成部分的具体组成与运行原理不同,通常把控制器区分为硬布线控制器和微程序控制器两大类。本书分别介绍了这两种控制器,具体实例选自用于教学实践的教学计算机系统,最大限度地把课堂授课和教学实验联系在一起。

6.2 硬布线控制器

硬布线控制器,又称为组合逻辑控制器,与微程序控制器共同构成计算机通用的两大类控制器。硬布线控制器是早期计算机唯一可用的方案,当前在RISC结构的计算机、追求特高性能的计算机中也被普遍选用。它的基本运行原理,是使用大量的组合逻辑门线路,直接提供出控制计算机各功能部件协同运行所需要的控制信号。这些门电路的输入信号是指令操作码、指令执行步骤编码,或许还有其他的控制条件,输出信号就是提供给计算机各功能部件的控制信号。其优点是,形成这些控制信号所必需的信号传输延迟时间短,有利于提高系统运行的速度。其缺点是,形成控制信号的电路设计比较复杂,再用与、或、非等组合逻辑门电路把设计结果实现出来也相对烦琐,尤其是要变动一些设计时不大方便。随着大(超大)规模集成电路的发展,特别是各种不同类型的现场可编程器件的出现,性能杰出的辅助设计软件的应用,这一矛盾已在很大程度上得到缓解。

6.2.1 硬布线控制器的组成和运行原理简介

硬布线控制器的基本组成如图6.2所示,由图可见,它与图6.1中的内容是一致的,程序计数器(PC)、指令寄存器(IR)都保持不变,只是指令执行步骤标记线路明确为节拍发生器(Timing),控制信号产生或保存电路明确为控制信号产生部件(CU)。

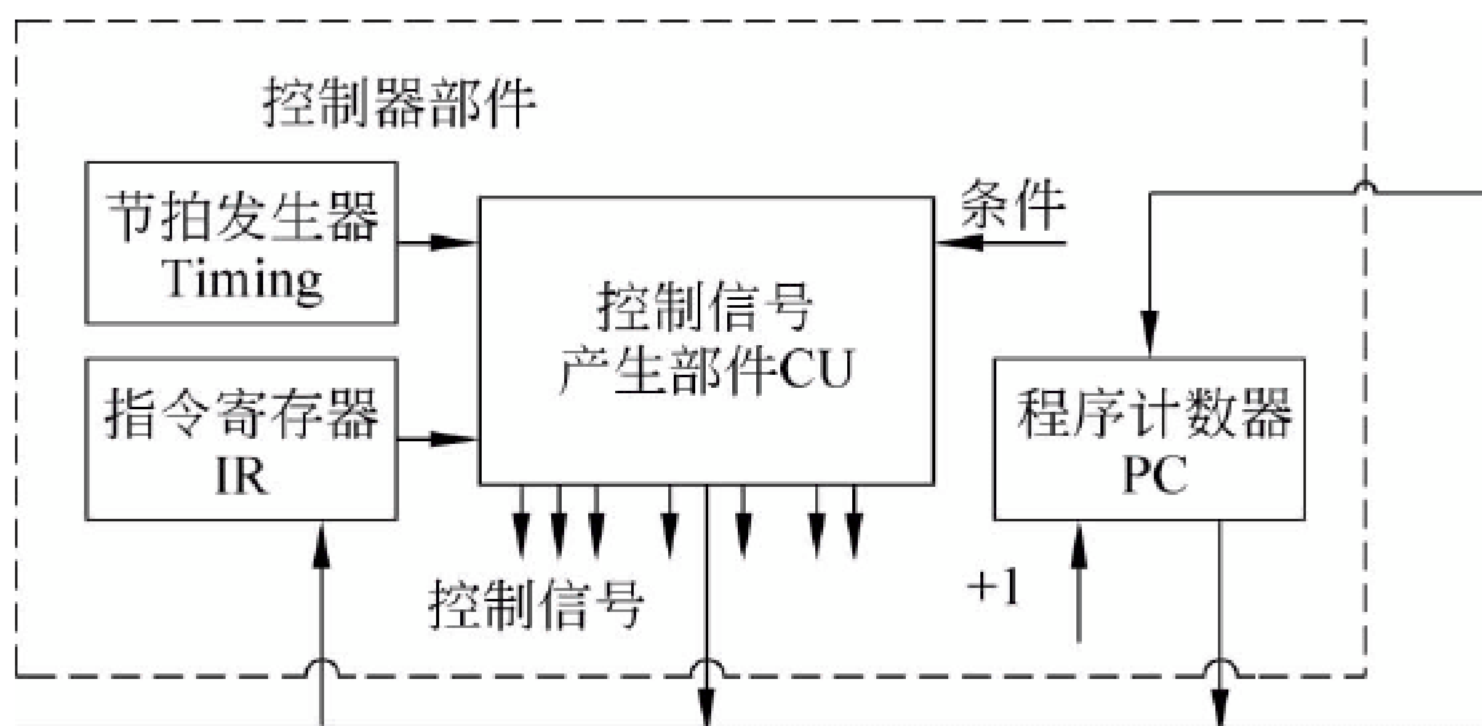


图6.2 硬布线控制器的组成

可以从以下几个方面来理解硬布线控制器的运行原理。

首先,从计算机**执行程序**的层次考虑。程序是机器指令的一个序列,因此,计算机硬件应能**自动地**、**连续地**执行程序中的每一条指令,核心问题在于应依据指令的执行次序,自动地逐条从内存储器读来每一条指令,接着分析和执行这条指令,**程序计数器**(PC)在这一过程中起到关键作用。它保存一条指令在内存中的地址,服务于到内存中读取指令。它的自行增量功能用于形成相邻的下一条指令的地址,解决的是指令的顺序执行需求,而通过接收新的指令(例如转移指令)的地址来改变指令的执行顺序,解决的是变更指令执行流程(非顺序执行)的需求,这两种办法确保计算机能**自动地**、**连续地**执行程序中的每一条指令。

从内存储器读出来的指令内容将经过数据总线传送到**指令寄存器**(IR),以便在这条指令执行的整个过程中,由IR来提供这条指令的主要内容(指令操作码和形式地址等)。

其次,从计算机执行一条指令的层次考虑。每条指令都是在取指、分析、执行的循环中完成的,即执行每一条指令,通常都要经过读取指令、分析指令、执行这条指令规定的具体操作功能等几个操作步骤。设计硬布线控制器或微程序控制器,都要经过如下几个步骤。

(1) 熟悉需要实现的每一条指令的功能和格式,了解完整的指令系统。

(2) 划分每一条指令的执行步骤,把每一条指令的完整功能拆分到对应指令的每一个执行步骤中去,确定这一步骤会用到哪些部件的什么功能。

(3) 确定在实现这些操作功能时,计算机各功能部件要求使用哪些控制信号。

(4) 需要选用什么具体逻辑线路,采用什么处理方案,来分步骤地形成并向计算机各功能部件提供出这些控制信号。

在完成划分指令执行步骤的设计之后,需要有办法区分、表示这些执行步骤。在硬布线控制器中,使用**节拍发生器**(Timing)来区分指令不同的执行步骤。它是由几个触发器电路实现的典型的时序逻辑电路,提供指令每一个执行步骤的节拍状态信号,用节拍状态变换来标明一条指令的执行步骤的次序关系。

在确定计算机各功能部件要求使用哪些控制信号时,依据的是这些需要控制的功能部件本身的组成和运行的控制需求。例如,要控制第4章讲过的运算器完成某种运算功能,就得向其提供它所要求的几组多位的控制信号;对硬布线的控制器方案,就必须选用组合逻辑的门线路来形成并提供出全部的时序控制信号,这是由**控制信号形成部件**(CU)承担的。它依据正处在执行过程中的指令的操作码(保存在指令寄存器IR中),当前指令所处的执行步骤(由节拍发生器的节拍状态标记)和某个(些)判别条件(例如ALU运算结果是否为0)等作为输入信号,用**与一或两级组合逻辑门电路**直接、快速地形成本节拍用到的全部控制信号,并送到计算机的各功能部件。在这些信号的控制下,计算机各功能部件会完成预期的操作功能。顺便指出,在使用现场可编程的大规模集成电路实现控制器时,原来由**操作码译码器**完成的功能已经归并到这片大规模集成电路中去,往往看不到单独的操作码译码器电路。

若暂不考虑处理中断的有关问题,则控制器的基本组成和运行原理的内容已经介绍完了,若就止步于此,学生学到的只限于一些抽象知识,更深入的实用内容尚未见到,也难以了解学过的知识可以用到哪里,怎么去用,就如同远远地看到一座山,看到的只限于山的轮廓和大体形状,而不是更精妙多彩的美景,自然也就感觉不到游玩的乐趣,要找到乐趣,需要走近大山,亲自爬一爬,仔细观赏,细细品味,山的每一处美景都会紧紧抓住你的眼球,刺激到你的每一根神经,让你陶醉与感叹,不虚此行!学习计算机组成原理课程也是这个道理,基础知识、基本技术应该了解,如同远处看山,还需要知道这些知识可以用到哪里,选用什么技术将它实现出来,实现出来的计算机系统有什么功能,怎样去使用它等,这才是更加精彩的部分,乐趣所在,但需要花费较多的时间和精力,相当于爬山观景。为此下面给出2个典型计算机系统及其控制器部件的实例。

6.2.2 MIPS32计算机的控制器简介

1. MIPS32计算机的指令系统

MIPS32是32位字长、典型RISC结构的计算机,即指令、ALU、主存储器、地址总线和

数据总线等都是32位字长。该系统只使用3种非常规范的指令格式,如图6.3所示。MISP指令系统用到的寻址方式和指令条数少。在有的型号中,还提供一套16位字长的指令系统,以更方便地支持嵌入式系统的使用要求。对主存储器中的数据支持按字、半字、字节3种方式读写。指令在存储器中按字对齐方式存储,保证每条指令都被保存在一个存储字单元中。地址总线的高30位用于访问存储字,最低2位用于区分半字和字节,按字访问存储器时,地址总线的最低2位的值为00。



图6.3 MIPS机的3种指令格式

第1种指令称为**R型指令**,完成rd←rs op rt的运算功能(ADD,SUB,AND,OR);rd←(rs＜rt)的操作功能(SLT),rs的内容比rt的内容小则rd置“1”,否则清0。

第2种指令称为**I型指令**,完成rt←rs op immediate的运算(ADDI,ANDI,ORI);rt←memory[base+offset]的操作功能(LW);memory[base+offset]←rt的操作功能(SW);若rt=rs,转移到地址PC+offset * 4,否则顺序执行;若rt＜＞rs,转移到地址PC+offset * 4,否则顺序执行。

第3种指令称为**J型指令**,完成无条件跳转的操作功能。例如J,即用PC的最高4位的值拼接target×4的值作为跳转的指令地址。

从对全部指令都选用相同的执行时间完成,还是为不同类型的指令设定不同的执行时间来区别,可以把计算机划分为单周期CPU系统、多周期CPU系统和流水线CPU系统3种实现方案。单周期CPU系统的硬件资源利用率和指令执行速度都比较低,不是很实用;多周期CPU系统在硬件资源利用率和指令执行速度两个方面都得到明显改善;流水线CPU系统的硬件资源利用率和指令执行速度最高,成为目前被广泛选用的方案。多指令周期方案是学习与理解指令流水线的必要准备,是计算机组成原理课程的重点教学内容,因此将以MIPS计算机的多周期CPU系统为例,介绍控制器的组成和指令执行过程,这里给出的内容更多地用于拓展学生的视野。

2. 多周期CPU系统的设计和实现方案

在多周期CPU系统中,指令需要几个时钟周期就为其分配几个周期,而不是所有指令使用相同的执行时间,有利于提高指令的执行速度和计算机硬件部件的利用效率。例如,功能最简单的指令只用2个步骤完成,另外几条指令用3个步骤,算术和逻辑运算指令和写内存储器指令用4个步骤,仅有读内存储器指令才用5个步骤完成。图6.4和图6.5给出了多周期CPU的MIPS计算机的逻辑框图和指令执行步骤(周期),对图中内容作如下说明。

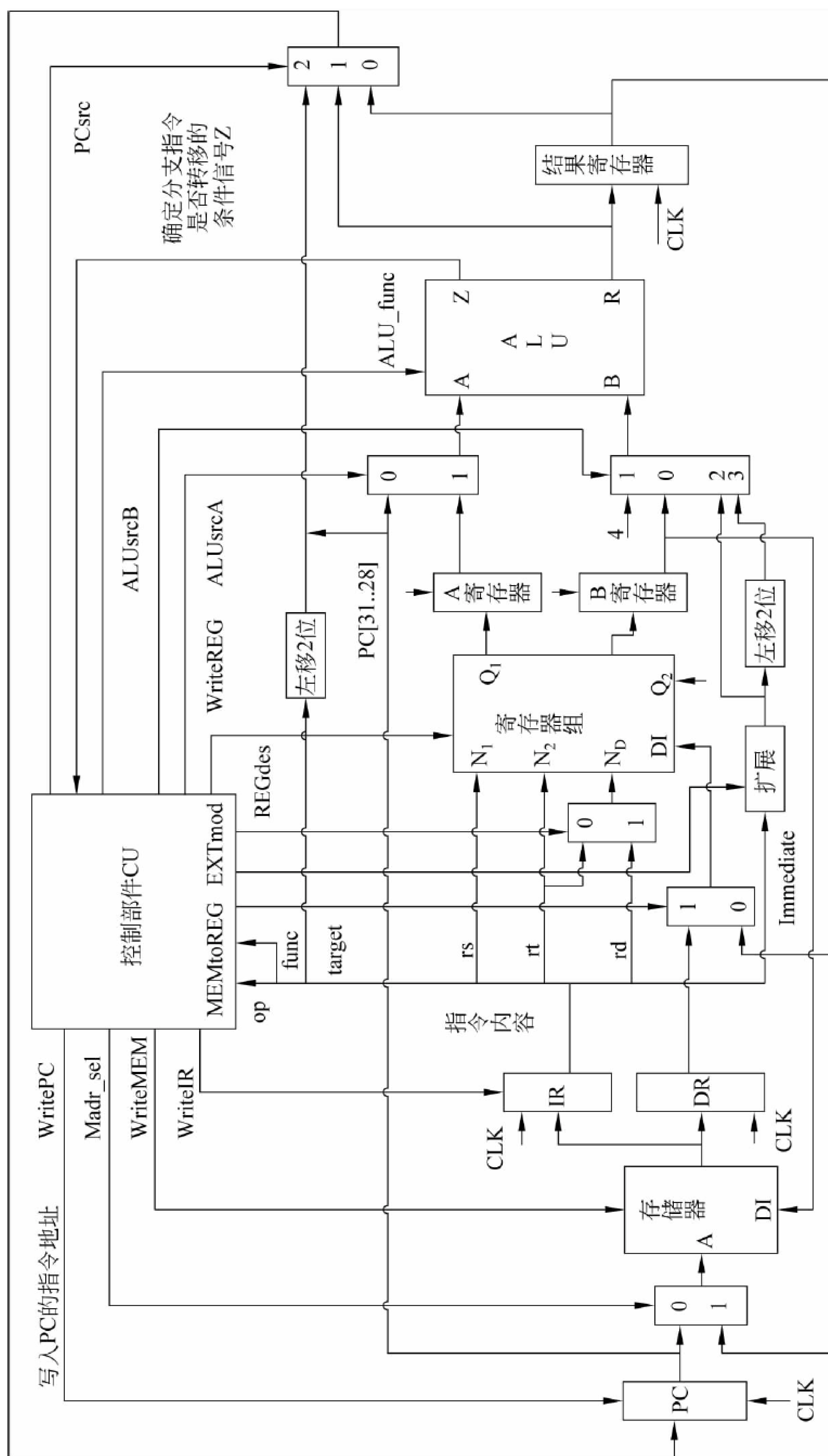


图 6.4　多周期 CPU 方案的 MIPS 计算机系统的组成

在给出某条指令在存储器中的地址时，只需要给出地址的高30位的值即可，最低2位的值必定为00，在给出某个数据在存储器中的地址时，则要给出地址全部32位的值。因此，在计算存储器单元(字、半字、字节)地址时，将如何使用指令中给出的16位的立即数(immediate)，针对读写数据和读指令有2种不同的处理。

(1) 在计算读写数据使用的地址时，这个16位的数是用补码表示的，它的最高一位是数的符号位，需要用这个符号位的值填充更高的16位(这项操作被称为符号扩展)，以便形成一个完整的32位的数据，才可以和另外一个保存在寄存器中32位的数据相加，得到的计算结果用于读写存储器的一个字、半个字或一个字节的地址。

(2) 读写指令时，必须以字为单位执行读操作，指令中给出的立即数也是16位的补码数，需要进行符号扩展并左移两位再和一个32位的指令地址执行加运算，使指令按字转移的范围扩大4倍。

(3) 对跳转指令J，指令中给出26位的伪直接地址，与上面介绍的处理立即数字段的含义类似，需要将其左移2位后拼接在PC的最高4位(PC[31..28])的右侧形成一个完整32位的指令地址。

此外，在使用立即数进行算术或逻辑运算的指令中，要区分是算术还是逻辑运算，对16位的立即数进行不同的扩展处理。

(1) 对带符号的算术运算，指令中给出的立即数是16位补码数，要进行符号扩展。

(2) 对逻辑运算，指令中给出的立即数是16个二进制表示的逻辑值，要用16个0填充最高16位的值，这一操作被称为0扩展。

在多周期CPU的计算机系统中，可以使用一个存储器既存放指令又存放数据，因为每条指令的功能是在几个周期中完成的，例如读取指令只在取指周期(Sif)执行，读取数据要到数据读写周期(Smen)执行，时间上是错开的，不会产生冲突。但是需要注意，为读取指令和读写数据可能使用不同来源的地址，读取指令时使用的地址来自于程序计数器(PC)，读写数据时使用地址来自于ALU的计算结果(接在ALU输出端的寄存器C)。为此需要在存储器的地址输入端A使用一个2选1的电路选择2路地址来源；读出来的信息也要保存到不同的线路中，指令需要保存在指令寄存器(IR)中，数据将保存在存储器的数据寄存器(DR)中。写入存储器的数据总是来自于寄存器堆的输出Q_2，通过寄存器B送到存储器的数据输入端(DI)。

在多周期CPU的计算机系统中，可以把计算下条相邻指令地址和计算指令转移地址这两项运算功能都交由ALU部件完成，在取指周期Sif完成PC+4运算，在译码周期Sid完成对已经加过4的PC值和执行过符号扩展并左移了2位后的立即数的求和运算并保存结果到结果寄存器C中。为此需要对参加ALU运算的两路数据进行选择，最终使ALU的A路输入变成现在的2选1(选择寄存器A或PC)，B路输入变成现在的4选1(选择寄存器B、立即数经符号扩展或0扩展的结果、常数4、立即数经符号扩展并左移了2位的结果)。对算术逻辑运算指令，ALU可以完成A op B、A op immediate运算；在计算指令地址时，PC可以接收3个来源的结果，用一个3选1的选择器电路完成选择。

(1) 对指令顺序执行，PC接收PC+4的运算结果(ALU的直接输出)，在取指周期Sif完成；

(2) 对J指令，PC接收PC(经过加4运算)的最高4位的值拼接指令中的target左移2

位后的值，在指令译码周期 Sid 完成。

（3）对 BEQ 和 BNE 指令，若转移条件成立则转移，PC 接收保存在寄存器 C 中的转移地址，在执行周期 Sexe 完成，若条件不成立，PC 不接收转移地址，使用 PC+4 的值顺序执行。

在多周期 CPU 的系统中，从寄存器组读出来的两路数据都要缓冲在两个专用的寄存器 A 和 B 中，写入的数据同样来自两处来源，ALU 的结果寄存器 C 和存储器的数据寄存器 DR，需要使用 2 选 1 的选择器电路。

在多周期 CPU 的计算机系统中，不同类型的指令使用不同个数的时钟周期，每条指令的每个执行步骤被安排在不同的时钟周期中完成，需要控制部件依据正在执行的指令和指令当前所处的执行步骤向计算机各部件提供所要求的控制信号。例如，至少可以为指令的 5 个执行步骤分配 5 个周期状态，如图 6.5 所示。

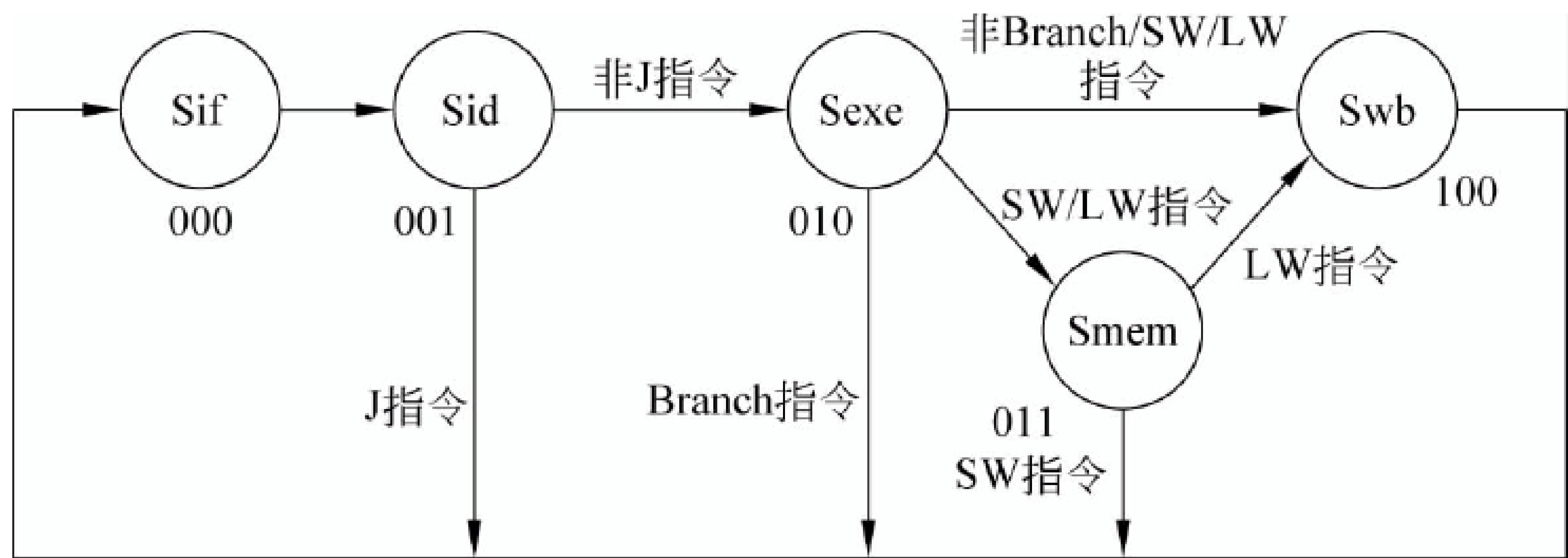


图 6.5　多周期 CPU 中的周期状态转移图

在状态图中，每一个圆形图代表一个状态周期；圆内的文字表示周期的名称，它们分别是取指周期、译码周期、执行周期、存储器读写周期、数据写回周期；圆外侧的数字表示状态编码；在周期之间的矢量线表示状态转移的方向；矢量线旁边的文字表示状态转移的控制条件，通常是指令类别。

从状态图可以看到，J 类型的指令只经过 Sif 和 Sid 两个周期就完成执行过程；而转移指令要经过 Sif、Sid 和 Sexe 这 3 个周期才完成执行过程；SW 还要经过 Smem 周期、算术逻辑运算指令经过 Swb 周期，它们各自用 4 个周期完成执行过程，仅有 LW 指令需要经过全部 5 个周期结束执行过程。

下面按照指令类别和它们不同的执行步骤，用表格和文字说明的方式，给出不同指令在各周期需要完成的功能，以及周期之间的衔接关系，如表 6.1 所示。

表 6.1　各类指令在不同周期需要完成的功能

<table>
<tr><th>周期指令类</th><th>取指令(IF)</th><th>译码(ID)</th><th>执行(EXE)</th><th>访存(MEM)</th><th>写回(WB)</th></tr>
<tr><td>J 指令</td><td rowspan="5">IR←Mem[PC]
PC←PC+4</td><td>PC←PC[31..28] ‖ (target<<2)</td><td></td><td></td><td></td></tr>
<tr><td>BRANCH 类</td><td rowspan="4">C←PC+(符号扩展(imm)<<2)
A←Reg[rs]
B←Reg[rt]</td><td>若条件为真，则 PC←C</td><td></td><td></td></tr>
<tr><td>R 类</td><td>C←A OP B</td><td></td><td>Reg[rd]←C</td></tr>
<tr><td>SW 指令</td><td rowspan="2">C←A + 符号扩展(imm)</td><td>Mem[C]←B</td><td></td></tr>
<tr><td>LW 指令</td><td>DR←Mem[rt]</td><td>Reg[rt]←DR</td></tr>
</table>

(1) 在取指周期(Sif)完成读取指令的功能,即用 PC 作地址完成从存储器中读出指令并将其保存到指令寄存器 IR 中,同时完成 PC 加 4 的操作,得到的是下一条相邻指令的地址。这项功能公用于所有指令,对所有指令使用完全相同的控制信号,结束后无条件进到 Sid 周期。

(2) 在译码周期(Sid)完成指令译码的功能,所有指令都要在本周期完成相应的操作。完成后将依据不同指令进入不同的后续周期。请注意,此周期对 J 型和非 J 型指令执行的功能是不同的,**对 J 指令**,把通过 PC[31..28]拼接(target<<2)得到的跳转地址存入 PC,就结束本指令的执行过程,返回取指周期,进入下一条指令的执行过程。**对非 J 型指令**,则执行从寄存器组读出 2 路数据并分别保存到寄存器 A 和 B,为指令下一步运算准备源数据,同时执行 PC 内容与经过符号扩展并左移 2 位的 Immediate 的相加操作,求得跳转指令的地址并保存到结果寄存器 C 中,之后进入执行周期(Sexe),开始非 J 类指令的下一步操作。

(3) 在执行周期(Sexe),不同指令将执行不同的计算处理功能,结束后将转移到不同的周期,有 3 种情形。**对 BEQ 和 BNE 指令**,如果转移条件成立则应该执行转移,需要把前一周期保存在结果寄存器 C 中的转移地址存入 PC,否则保持 PC 原有内容不变,之后结束本指令的执行过程,返回取指周期,进入下一条指令的执行过程。**对算术逻辑运算指令**,ALU 要完成相应的计算并保存结果到寄存器 C,之后转移到结果写回周期(Swb)。**对内存储器读写指令**(LW/SW),ALU 要完成加法运算求出要用到的内存地址并保存结果到寄存器 C,之后进入内存储器读写周期(Smem)。

(4) 内存储器读写周期(Smem)执行内存读写操作,内存地址在前一个周期已经保存在结果寄存器 C 中。**对写存储器指令**,通过发写内存命令,把从寄存器堆读出来、已保存在寄存器 B 中的数据写入存储器,完成后结束本指令的执行过程,返回取指周期,进入下一条指令的执行过程;**对读存储器指令**,通过发读内存命令,启动读存储器操作并把读出内容保存到数据寄存器 DR 中,然后进入结果写回周期(Swb)。

(5) 在结果写回周期(Swb),通过发写寄存器命令,把 ALU 的计算结果(已经保存在寄存器 C 中)或从存储器中读出的数据(已经保存在数据寄存器 DR 中)写回到寄存器组,之后结束这类指令的执行过程,返回取指周期,进入下一条指令的执行过程。

在多周期 CPU 中,控制部件把指令寄存器的操作码字段(op)、功能码字段(func)的内容和 ALU 提供的判 0 结果 Z 信号用作输入信息,产生并送出 12 组(合计 18 位)的控制信号,可以划分成 4 类汇总如下。

(1) **用于 PC 和 IR** 的控制信号。PCsrc[1..0]选择写入 PC 的 3 个指令地址,00 选择指令顺序执行地;01 选择指令分支执行地址;10 选择指令跳转执行地址。

WritePC 是 PC 的写入命令;WriteIR 是 IR 的写入命令。

(2) **用于寄存器组**的控制信号。MEMtoREG 选择写入寄存器组的数据来源(0:寄存器 C;1:寄存器 DR);REGdes 选择写入的寄存器编号(0:rt;1:rd);WriteREG 是寄存器组的写入命令。

(3) **用于 ALU** 的控制信号。EXTmod 选择 Immediate 扩展方式,0 代表零扩展;1 代表符号扩展。ALUsrcA 选择送到 ALU 的 A 端的数据来源(0:PC+4 的结果,1:寄存器 A);ALUsrcB[1..0]选择送到 ALU 的 B 端的数据来源,00 代表寄存器 B;01 代表常数 4;10 代

表扩展的 imm 的值；11 代表扩展并左移 2 位的 imm 的值。ALU_func[4..0]用于选择 ALU 部件的运算功能，×0001 代表加运算；1001 代表减运算；×1010 代表比较运算；00000 代表与运算；01000 代表或运算。

（4）**用于存储器**的控制信号。Madr_sel 用于选择存储器地址的来源，0 选择 PC；1 选择寄存器 C。WriteMEM 是存储器的写入命令。

6.2.3 TEC_XP_Ⅱ教学计算机的硬布线控制器的设计与实现

这台教学计算机是本书作者专门针对计算机硬件教学，特别是计算机组成原理课程的教学与实验而设计实现的。其特点是组成简单完整，体现原理清楚，紧密配合教材，易于动手操作，配备小巧软件系统，汇集了此前二十多年间研发的多种机型的特点和长处，又在设计目标、设计和实现的手段等方面作了重大改动，着力增强实验操作的方便程度。下面分层次地讲解这个控制器部件的有关内容。

1. 教学计算机的部件组成及其连接关系

控制器部件用于控制整机系统中的各个部件协同运行。讲解控制器之前，一定要对整机系统有个初步了解，这里首先把实现这台计算机的指令系统、各部件的功能与组成，以及上述内容都出现在教材的哪一章节说明清楚。从不同的教学进度安排考虑，有些章节可能已经讲过，这里只是把那一章中的部件实例拿过来构建整机系统，而另外一些章节可能尚未讲到，详细内容学生还不是太懂，需要教师把构建整机系统用到的少量知识进行简单介绍，例如把内存储器、总线、接口和输入输出设备的完整内容安排在控制器教学之后再学习就是可行的选择。若把控制器教学实验安排在全部授课结束后的期末进行，在开展控制器部件、几个执行部件实验的时候就会遇到一些难以调和的矛盾。

TEC_XP_Ⅱ实现的是教学计算机的**指令系统**，相关内容取自本教材 5.3.4 节的指令系统实例，在硬件系统保持与前期产品相同功能的基础上，又确保了良好的软件兼容性。

这台计算机的**运算器部件**取自本教材 4.2.3 节的运算器实例；**控制器部件**是本节讲解的控制器部件实例；**存储器部件**取自本教材 7.3 节的存储器部件实例；**串行接口**取自本教材 11.3.3 节的接口实例；总线采用本教材 11.2.2 节讲述的**单总线结构**；输入输出采用本教材 11.4.1 节讲述的**程序直接控制方式**；**输入输出设备**是通过 PC 实现的仿真终端，只要运行 PC 系统中的 pcec16.com 程序，PC 就成为教学计算机的输入输出设备，使教学计算机有了执行字符型数据的输入输出能力。

现在的首要工作是设计教学计算机整机系统的组成，之后再来实现这个控制器部件。整机系统的设计结果如图 6.6 所示，还可以进一步抽象，简化为图 6.1 的表示方式。

实验计算机字长 16 位，寄存器、总线、指令和数据通常都选定为 16 位字长，存储器按字寻址，指令系统与 PC8086 的指令系统比较接近，但更为简单精小。

从图 6.6 中可以看到，教学计算机硬件系统包括了计算机传统的 5 大功能部件：控制器部件（在图的左上部），运算器部件（在图的左下部），内存储器部件（在图的右下部），输入设备（键盘）和输出设备（显示器），输入输出设备是用 PC 实现的仿真终端（在图的右上方），经电平转换电路 MAX202 芯片接入主机系统。

本系统采用的是单总线结构，通过把全部的功能部件都直接连接到这组总线上来组成整机系统，部件之间需要经过总线完成信息传送。

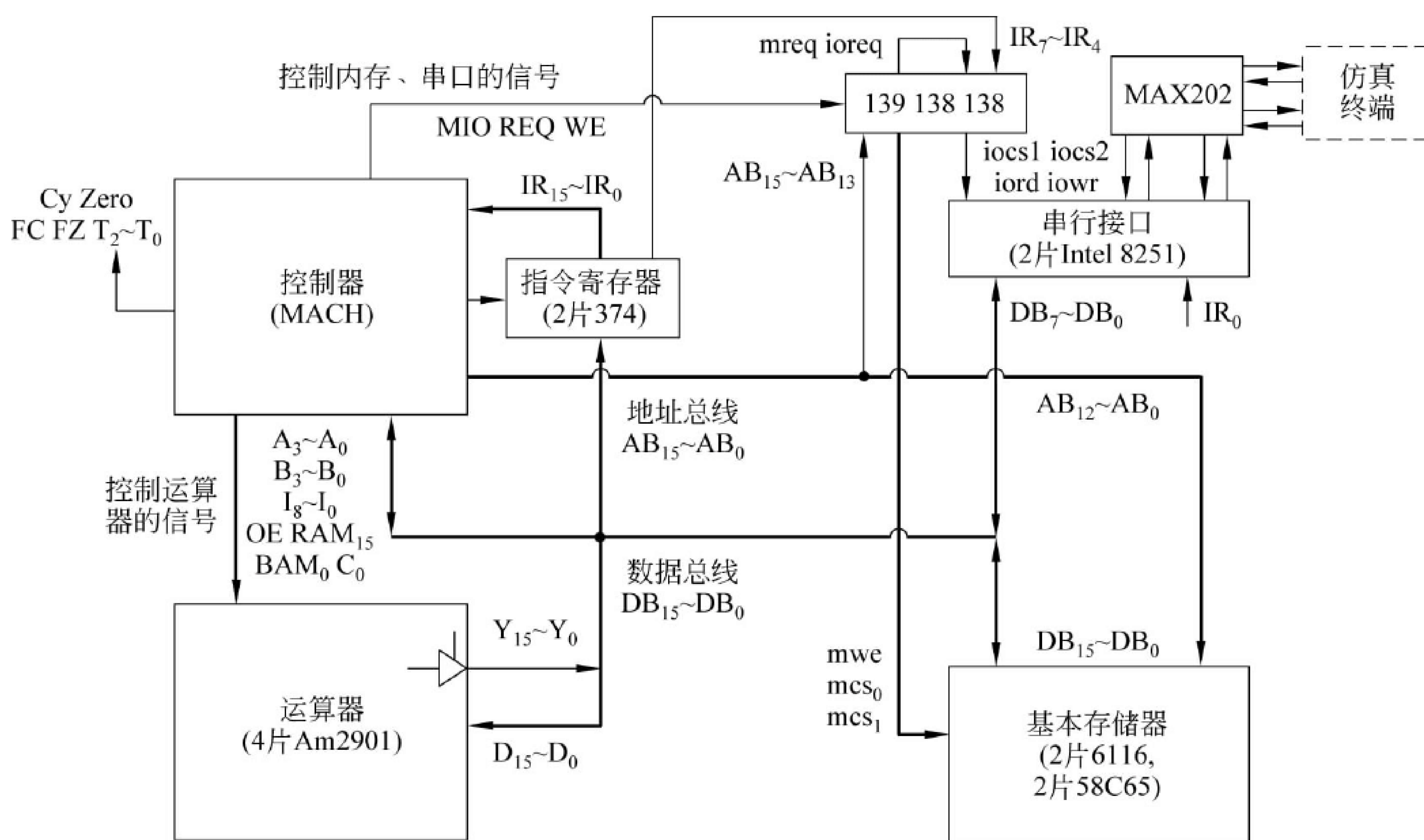


图 6.6　实验计算机的整机系统的基本组成框图

(1) **数据总线 DB** 直接连接到存储器和串行接口芯片双向输入输出的数据线引脚，又连接到运算器芯片的数据输入引脚 D 和带有三态输出控制的数据输出引脚 Y，还连接到控制器部件的(MACH 芯片)的信息输入引脚和输出引脚(双向输入输出，输出带有三态控制)，以及指令寄存器 IR 的输入引脚，这样就可以支持主机的几个部件之间正常的数据传送功能。

(2) **地址总线 AB** 直接连接到控制器部件的地址输出引脚，其最高 3 位连接一片 138 译码器用于产生存储器的片选信号，最低的 13 位送到存储器芯片的地址线引脚，用于选择存储器芯片内的一个存储单元。IO 端口地址直接由指令寄存器 IR 的低 8 位提供。

(3) **控制总线 CB** 通常不会表示在整机系统的组成框图中，我们却在图 6.6 中以非常醒目的方式表明了控制器部件对运算器、内存和串口提供的控制信号和少量数据信息，包括为控制运算器部件使用了 17 位的控制信号，还用到 4 位的数据和标志位信息，传送这 21 位信号不必使用控制总线；为控制内存和串口只用到 3 位的控制信号(通过控制总线传送)，再经由 3 片译码器芯片产生内存、串口的读写命令和片选信号(通过控制总线传送)。

在具体深入地学习控制器部件的组成和功能之前，先概要了解这些内容是必要的。

2. 教学计算机的线路组成

整机硬件系统的线路组成如图 6.7 所示。在这张图中，比较详细地给出了控制器部件的线路组成，学懂控制器的功能与组成是计算机组成原理课程的重点教学内容。

硬布线控制器部件由 1 片 ispMACH 芯片和 2 片 74LS377 芯片组成，运算器部件由 4 片 4 位的 Am2901 运算器芯片组成，这 2 个部件合在一起构成 CPU 系统。

内存储器部件由 2 片 8 位的 ROM 芯片(用于保存监控程序)和 2 片 8 位的 RAM 芯片组成(用于保存数据和用户程序)。

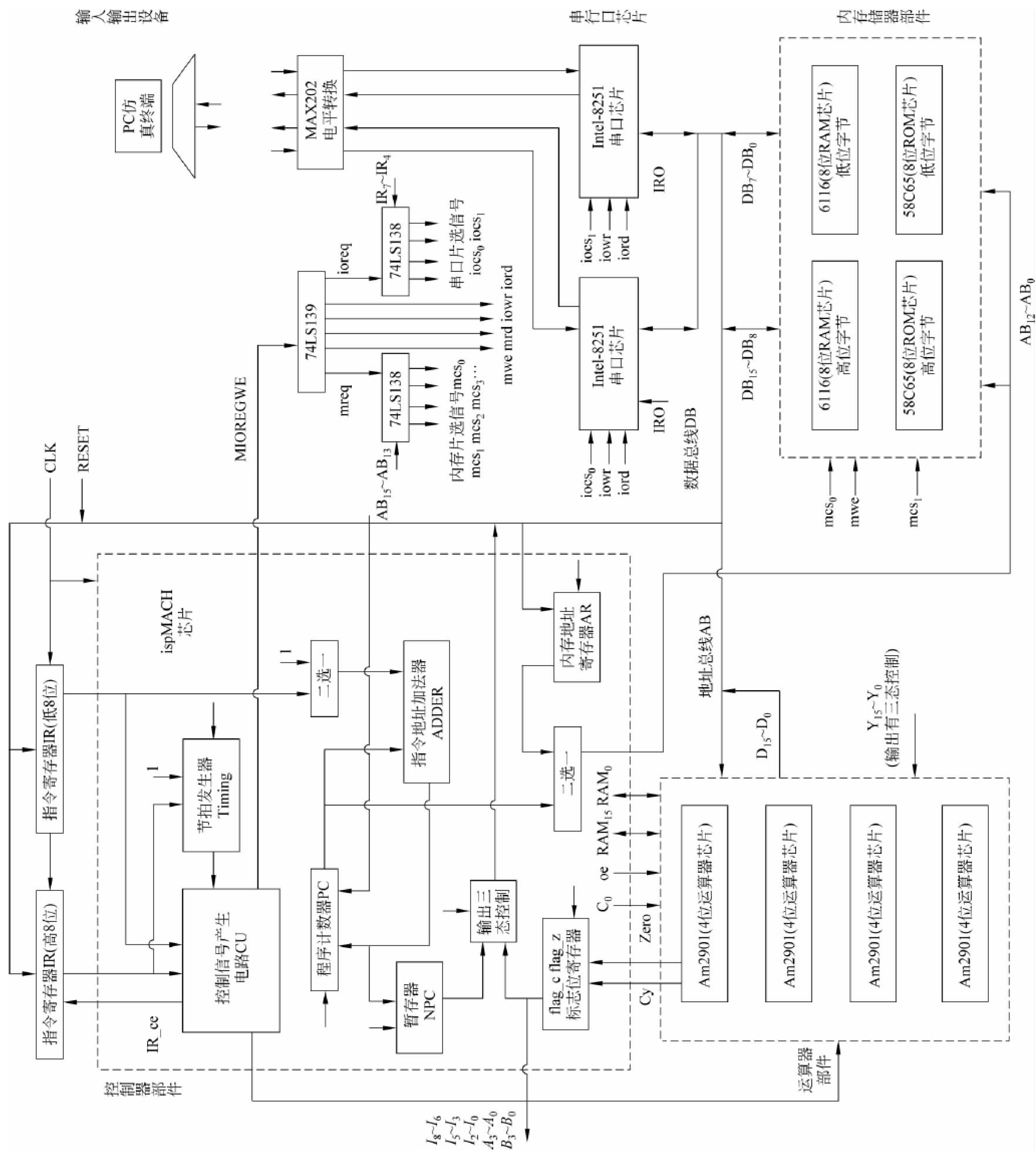


图 6.7 教学计算机的整机系统组成和线路实现框图

2 路串行接口选用的是 Intel-8251 芯片(在内存储器部件上方),用于连接 PC 实现的仿真终端,支持字符型数据的输入和输出操作,以便支持和运行监控程序。

在这个硬件系统中,运算器、存储器、串行接口电路选用的都是功能固定的芯片,核心知识是这些芯片的内部线路组成和运行原理,涉及的技术是用不同芯片构建计算机的部件;只有 isp MACH 芯片是现场可编程的,主要用于实现控制部件的 PC、CU、Timing 等相关的功能,而把控制部件中的 IR 设置在这个芯片之外,选用 2 片 74LS377 芯片来实现。我们还把内存地址寄存器 AR、标志位寄存器 flag、与中断有关的电路(图中未画出)也一并纳入到这个芯片中实现。

在这个芯片内,还实现了一个 16 位加法器,专用于计算指令地址(PC+1 或 PC+offset)。

仿真终端是通过执行 PC 系统中的 pcec16.com 程序启动运行的,同学应该更多地关注它实现的功能和操作方法。

从图 6.7 中还可以看到,系统中还使用了 3 片译码器芯片,1 片双二-四译码器 74LS139 通过对 CPU 提供出来的 mio、req、we 这 3 位信号的译码,产生存储器与串行口的读写命令(mwe、iord、iowr),2 片三-八译码器 74LS138 分别用于对地址总线的最高 3 位、指令寄存器的 IR_7～IR_4 进行译码,产生内存和串口芯片的片选信号(mcs_0、mcs_1、mcs_2、…、$iocs_0$、$iocs_1$)。

还有 1 片 MAX202 芯片,用于实现教学计算机串行口的输入输出信号的电平转换,确保使其能够与仿真终端那一方的串行口进行正常连接和通信。

3. 控制部件的组成、设计与实现

此前已多次说到,在教学计算机中,硬布线方案的控制器部件需要由程序计数器 PC、指令寄存器 IR、节拍发生器 Timing、控制信号产生电路 CU 这 4 个子部件组成。它的 IR 选用 2 片 8 位的寄存器 74LS377 芯片构成,其余 3 个子部件都在可编程的 ispMACH 芯片内部实现。

4. 设计指令系统

为教学计算机设计的指令系统已在教材的第 5 章讲解过,并使用汇编语言开展了必要的程序设计实验。现在只把它的指令格式复制在这里,如图 6.8 所示。具体的指令请查阅教材的 5.3.4 节内容。

<table>
<tr><td rowspan="2">操作码</td><td>DR</td><td>SR</td></tr>
<tr><td colspan="2">IO 端口地址 / 相对偏移量</td></tr>
</table>

<table>
<tr><td>立即数 / 直接内存地址 / 变址偏移量</td></tr>
</table>

图 6.8　教学计算机的指令格式

5. 划分指令的执行步骤

划分节拍的依据:若硬件上能实现,以用更少的步骤完成指令的运行过程为好,为此可为不同指令选用不同的步骤数,这被称为多种指令周期方式;读取指令用一个步骤;完成数据运算用一个步骤,读或写一次数据存储器用两个步骤(分别完成送地址到地址寄存器

AR，执行存储器的数据读写操作）。

可以看到这种划分结果中没有用到分析指令的步骤，这是因为这台教学计算机的组成很简单，强调突出 CPU 的基本组成和指令的顺序执行过程，没有使用提高指令执行速度的更复杂的技术。

对于单字指令，读取指令可以在一个步骤中完成，这是因为 PC 的内容（指令地址）可以直接送到（不必经过内存地址寄存器 AR）地址总线 AB；对于双字指令，读取指令的两个指令字各用一个步骤完成。而读写数据则要用两个步骤。

运算器完成一次数据运算可以用一个步骤完成，因为在使用的运算器芯片中，把读寄存器组中的数据、ALU 运算、结果写回寄存器这 3 项操作安排在同一个时钟周期完成。还需要注意，本硬件系统中没有设置存储器的数据寄存器，从主存、串口中读出的数据信息可以直接经过数据总线 DB 写入到运算器的某个寄存器中，这与 MIPS32 系统有明显不同。

本控制器选用多指令周期方式实现，若暂不考虑中断响应和处理功能，全部 30 条基本指令都可以在 2～4 个周期完成，其中的 21 条 A 组指令用 2 个周期完成，8 条 B 组指令用 3 个周期完成，仅有 1 条 CALA 指令用 4 个周期完成（包括读取双字指令的 2 个指令字用 2 个步骤，保存程序断点到堆栈用 2 个步骤，是一次写数据存储器的操作）。节拍的划分结果如图 6.9 所示。



图 6.9　30 条基本指令的 4 个执行步骤及其衔接关系

（1）**取指周期**是每条指令的第一个执行步骤，完成从存储器中读取指令并存入 IR（MEM→IR）和形成下条指令地址（PC+1→PC）的功能，不受指令控制。要用到内存和控制器。

（2）**执行周期**是全部 30 条指令的第二个执行步骤，不同指令将完成不同的操作功能。

① 12 条算术和逻辑运算指令，要**在运算器部件内部完成**相应的计算操作，依据指令要求保存计算结果，还要保存标志位信息到 flag 寄存器（设置在控制器芯片中）。

② 5 条相对转移指令，要**在控制器部件内部完成**相应的转移地址计算，依据指令要求保存转移地址到 PC。

③ 2 条双字指令中的长转移指令，要从存储器读取指令转移地址（第二个指令字）并送 PC；对传送立即数指令，要从存储器读取立即数（第二个指令字）传送到运算器的 DR 寄存器，执行 PC+1 并送 PC。要用到内存和控制器 / 运算器。

④ 2 条输入/输出指令，要在运算器的 R_0 寄存器和串行接口之间完成 8 位数据的传送。要用到运算器和串口。

⑤ 8 条需要读写数据存储器（含堆栈区）的指令，要计算存储器的单元地址并传送到存储器的地址寄存器 AR。要用到运算器和控制器芯片中的 AR。

⑥ 1 条 CALA 指令要从存储器读子程序入口地址（第二个指令自）送 PC，PC+1 暂存到 NPC。

(3) **存储周期_1** 是9条指令的第三个执行步骤,其中3条指令完成写各自数据到存储器,另外5条指令完成从存储器中读出信息并写入各自指定的寄存器。要用到内存和运算器、控制器。CALA指令需完成修改堆栈指针并传送结果到AR寄存器。要用到运算器、控制器中的AR。

(4) **存储周期_2** 是CALA指令的第四个执行步骤,完成写NPC内容(主程序断点)到堆栈。要用到内存和控制器中的NPC。

6. ABEL程序实例

控制器的主要功能都在可编程的MACH芯片内部实现,需要通过ABEL-HDL硬件描述语言来描述其电路组成和功能,为此会涉及这个语言的语法规则、程序结构等内容,以及对ABEL程序的编译、优化,对结果的下载等操作方法。我们会以适当手段提供必要文档说明,但不准备在这里对这些内容进行详细讲解,还是直接给出一个最简单的实现4条指令的ABEL程序实例更好。

下面给出的是实现4条基本指令(ADD、AND、MVRD、JMPA)的硬布线控制器的ABEL语言源程序,程序有60多行并加了详细注释。学生需要认真学习研究这个程序的内容,之后才会有能力在这个程序中增加属于算术逻辑运算及相对转移的3条指令(SUB、SHR、JRNC)、内存读写的2条指令(LDRR、STRR)、执行输入输出的2条指令(IN、OUT),作为学习控制器、使用存储器、使用接口的实验内容。使用这11条指令写出的小程序,可以控制整机系统各部件协同运行。

```
MODULE TEC_new"
TITLE  'controller component'"
"2014/10/18
DECLARATIONS
RESET,CLK         pin 151,68;
Cy,Zero,ram0      pin 169,171,139;
IR15..IR0         pin 64..57,54..47;
AB15..AB0         pin 87..80,77..70;
DB15..DB0         pin 24,23,26,25,28,27,30,29,32..39;
MIO,REQ,WE        pin 95,94,93 istype 'com';
I8..I0            pin 14..21, 135 istype 'dc,com';
B3..B0,A3..A0     pin 9..12, 5..8 istype 'com';
aluoe,c0,ram15    pin 136,141,137 istype 'dc,com';
T2..T0            pin 162,160,158;
F_C,  F_Z         pin 168,170;
IR_G,IR_clk       pin 174,175;
pc15..pc0                           node istype 'reg,keep';
ar15..ar0                           node istype 'reg,keep';
t_2..t_0,flag_c,flag_z              node istype 'reg,keep';
pc_oe, ar_oe                        node istype 'com';
flag_c_oe,flag_z_oe                 node istype 'com';
A_,B_                               node istype 'dc,com';
sum15..sum0,jr_zu                   node istype 'com';
wk15..wk0,cy15..cy1                 node istype 'com';
```

```
c,z,x= .C.,.Z.,.X.;
pc= [pc15..pc0];                    ar= [ar15..ar0];
AB= [AB15..AB0];                    DB= [DB15..DB0];    sum= [sum15..sum0];
timing= [t_2..t_0];                 cif= (timing= = [0,0,0]);
cexe = (timing= = [0,1,0]);         cmeml= (timing= = [0,1,1]);
ir_op= [IR15..IR8];
ADD = (ir_op= = [0,0,0,0,0,0,0,0]);     AND = (ir_op= = [0,0,0,0,0,0,1,0]);
MVRD= (ir_op= = [1,0,0,0,1,0,0,0]);     JMPA= (ir_op= = [1,0,0,0,0,0,0,0]);
SUB = (ir_op= = [0,0,0,0,0,0,0,1]);     SHR = (ir_op= = [0,0,0,0,1,0,1,1]);
JRNC= (ir_op= = [0,1,0,0,0,1,0,1]);
LDRR= (ir_op= = [1,0,0,0,0,0,0,1]);     STRR= (ir_op= = [1,0,0,0,0,0,1,1]);
IN_  = (ir_op= = [1,0,0,0,0,0,1,0]);    OUT = (ir_op= = [1,0,0,0,0,1,1,0]);

EQUATIONS
  T2= t_2;      T1= t_1;      T0= t_0;
  F_C= flag_c; F_Z= flag_z;

[timing,pc,ar,flag_c,flag_z].clk= CLK;
pc_oe= RESET# !RESET& (cif# cexe& (MVRD# JMPA# JRNC&!flag_c));
when RESET then {pc= 0;   timing:= 0;} else
{ when cif then timing= [0,1,0];
when cexe& (STRR# LDRR) then timing= [0,1,1];
when pc_oe then
pc = sum& (cif# cexe& (MVRD# JRNC& !flag_c))
        # DB& cexe&JMPA;
} IR_G= !cif;   IR_clk= CLK;

  flag_c_oe = cexe& (ADD# SUB# SHR# AND); flag_c.oe= flag_c_oe;
when flag_c_oe then flag_c= cexe& ((ADD# SUB) &Cy# SHR&ram0# AND&0);
  flag_z_oe = cexe& (ADD# SUB# AND); flag_z.oe= flag_c_oe;
when flag_z_oe then flag_z= cexe& (ADD# SUB# AND) &Zero;

  ar_oe= cexe& (STRR# LDRR);   ar.oe= ar_oe;
when ar_oe then ar= DB;
when cmeml& (STRR# LDRR) then AB= ar; else AB= pc;

when ([A_,B_]= = [0,0]) then {[B3..B0]= [IR7..IR4]; [A3..A0]= [IR3..IR0];}
when ([A_,B_]= = [0,1]) then {[B3..B0]= [0,1,0,0];   [A3..A0]= [0,1, 0,0];}
when ([A_,B_]= = [1,0]) then {[B3..B0]= [0,0,0,0];   [A3..A0]= [0,0, 0,0];}
  ram15= 0;    ram15.oe= !IR7;

@ include 'adder_pc1.abl'
TRUTH_TABLE
([t_2..t_0,IR15..IR8]-> [aluoe,c0,I8..I6,I5..I3,I2..I0,A_,B_,MIO,REQ,WE])          "flag
[0,0,0, x,x,x,x,x,x,x,x]-> [1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 0,0,1]; "取指 MEM(pc)→IR,PC+1→PC
```

```
[0,1,0, 0,0,0,0,0,0,0,0]->[0,0, 0,1,1, 0,0,0, 0,0,1, 0,0, 1,0,0];  "ADD  DR+SR→DR  c,z
[0,1,0, 0,0,0,0,0,0,1,0]->[0,0, 0,1,1, 1,0,0, 0,0,1, 0,0, 1,0,0];  "AND  DR&SR→DR  0,z
[0,1,0, 1,0,0,0,0,0,0,0]->[1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 0,0,1];  "JMPA MEM(pc)→PC
[0,1,0, 1,0,0,0,1,0,0,0]->[1,0, 0,1,1, 0,0,0, 1,1,1, 0,0, 0,0,1];  "MVRD MEM(pc)→DR,PC+1→PC
“[0,1,0, 0,0,0,0,0,0,0,1]->[0,1, 0,1,1, 0,0,1, 0,0,1, 0,0, 1,0,0];  SUB  DR-SR→DR  c,z
“[0,1,0, 0,0,0,0,1,0,1,1]->[0,0, 1,0,1, 0,0,0, 1,0,0, 0,0, 1,0,0];  SHR  DR/2→DR  c,/
“[0,1,0, 0,1,0,0,0,1,0,1]->[1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 1,0,0];  JRNC ?PC+offset→PC
“[0,1,0, 1,0,0,0,0,0,1,0]->[1,0, 0,1,1, 0,0,0, 1,1,1, 1,0, 0,1,1];  IN_  (PORT)→R0
“[0,1,0, 1,0,0,0,0,1,1,0]->[0,0, 0,0,1, 0,0,0, 1,0,0, 1,0, 0,1,0];  OUT  R0→(PORT)
“[0,1,0, 1,0,0,0,0,0,0,1]->[0,0, 0,0,1, 0,0,0, 1,0,0, 0,0, 1,0,0];  LDRR SR→AR
“[0,1,0, 1,0,0,0,0,0,1,1]->[0,0, 0,0,1, 0,0,0, 0,1,1, 0,0, 1,0,0];  STRR DR→AR
“[0,1,1, 1,0,0,0,0,0,0,1]->[1,0, 0,1,1, 0,0,0, 1,1,1, 0,0, 0,0,1];  LDRR MEM→DR
“[0,1,1, 1,0,0,0,0,0,1,1]->[0,0, 0,0,1, 0,0,0, 1,0,0, 0,0, 0,0,0];  STRR SR→MEM
END
```

这个程序由4个基本段组成,包括头段、结束段、说明段、逻辑描述段,符合**ABEL**语言的规定。

1）头段

头段由程序开始的前3行组成,用标识符给出工程项目模块名,用字符串给出标题名,用注释指出程序的文件名。

2）结束段

结束段是程序的最后一个语句**END**,表明程序至此全部结束。

3）说明段

说明段是在**DECLARATIONS**之后给出的语句,用于描述以下3部分内容。

(1) MACH芯片的输入输出信号名称及其对应的器件引脚号,并以注释方式指出各自在逻辑电路中的用法或功能。

(2) 给出了MACH芯片的内部节点信号名称和类型,也以注释方式指出各自在逻辑电路中的用法或功能。

(3) 给出了程序中选用的专用常量和集合,表示节拍编码和指令的标识符等,这对于简化ABEL程序编写、提升程序的可读性非常有效。

4）逻辑描述段

逻辑描述段是在**EQUATIONS**之后给出的语句,描述了芯片内部基本电路的逻辑功能及其实现。程序中还以**@include 'adder_pc1.abl'**的方式把一段ABEL程序引入到本程序中,它实现的功能是**pc+1**或者**pc+offset**(带进位扩展)→pc,专用于计算指令地址,用到的只是线路设计知识,没有必要把这段程序直接写在本程序中。

逻辑描述段的核心部分包括两部分电路。

(1) 在**MACH**芯片内部的时序逻辑电路,包括程序计数器PC、内存地址寄存器AR、节拍发生器**Timing**、标志位触发器**flag_c**和**flag_z**,都是选用逻辑方程语句进行描述的,就是根据这些电路需要在哪一条指令(依据指令操作码)的哪一个执行步骤(依据节拍发生器编码)接收从哪里传送给它的信息或者需要送出信息到何处(哪个电路)去,来写出相应的逻辑方程式,这最为简捷和直观,从加了详细注释的ABEL程序中可以看得很清楚。

例如，程序中的 when RESET then {pc=0; timing:=0;} else {…} 语句表明了在系统复位时，timing 要被清零，之后每来一个时钟脉冲都应切换为一个新的节拍状态，状态转换关系是：000 状态时将无条件转换为 010 状态，在 010 状态时，若执行的是内存读写指令就转换为 011 状态，否则转回 000 状态。在这种运行情况下，需要通过用语句 timing. CLK=CLK 来为其指定时钟信号。

类似的是，程序计数器 pc 也需要在系统复位时被清零，但此后它只在某些条件下才接收新值，否则应保持不变，为此需要指定 pc 为带有接收使能控制的寄存器，这可以通过语句 pc. ce=pc_ce 完成，并且需要具体指出接收条件和 pc 接收的信息之间的对应关系，如：

```
when pc_ce then
pc= sum& (cif# cexe& (MVRD# JRNC& !flag_c))
# DB& cexe&JMPA;
```

pc_ce 是 pc 的接收使能控制，仅在该值为真时，pc 才接收输入；否则 pc 内容保持不变。pc 接收的内容有两个来源：专用加法器的输出 sum，或者是从内存读出来的指令地址（通过数据总线 DB 传送到 pc）。此处向 pc 赋值的赋值符使用的是"="，而不再是":="。为准确理解这条语句需要查阅有关 ABEL 语言的有关内容。

在本系统中，地址寄存器 AR 只用于接收并保存内存地址信息，并且需要在执行内存读写的前一个步骤完成，到了下一步再把 AR 的内容送到地址总线 AB，提供执行内存读写用到的地址信息。

请注意，到内存读取指令是不使用内存地址寄存器的，而是把保存指令地址的 PC 的输出直接送到 AB，使得读一个指令字能够在一个时钟周期完成。可见 AB 的信息需要在 PC 和 AR 二者中选用其一，我们是按照仅在读写数据内存时才送 AR 到 AB，其他情况都选 PC 送 AB。

标志位触发器的接收会用到 flag_c. ce、flag_c. clk、flag_z. ce、flag_z. clk 这 4 个语句。标志位触发器 flag_c，只在算术运算指令的 Cexe 周期才会接收 ALU 产生的 Cy 信号的值，在执行逻辑运算时应该使 flag_c=0，并规定移位指令的移位输出信号要保存到 flag_c。标志位触发器 flag_z，只在算术/逻辑运算指令的 Cexe 周期才会接收 ALU 产生的 Zero 信号的值。

(2) 在 MACH 芯片内的组合逻辑电路（主要是产生输出信号的 CU 电路），是选用真值表进行描述的，真值表的输入变量是节拍编码和指令操作码，输出变量是每一位输出控制信号，则真值表的每一行能准确清楚地表明是哪一条指令（依据指令操作码）的哪一个执行步骤（依据节拍编码），在真值表中的行、列交叉位置给出此时刻每一位控制信号的取值（0 或 1），这些信号将送到设备电路板上的运算器、存储器和串口，控制它们完成指定的功能。为了提高程序的可读性，在真值表的注释部分，给出了汇编语句名，本步骤需要完成的主要功能也包括如何维护标志位触发器等内容。

填写真值表的内容是控制器设计的核心工作，填写过程容易理解，即遵照 ABEL 语言的有关语法和格式，把指令执行流程表的内容转抄到这里的真值表中即可完成，工作变得简单轻松，无须设计者花费大量的时间和精力去设计每一位控制信号的逻辑方程式，而是把这项工作留给工具软件来自行完成。需要时设计者还可以随时查看到这些逻辑方程式，这对

找出并改正程序中的错误很有帮助。下面对这张真值表的内容进行详细讲解。

这张真值表的内容由5行组成,每一行的输入信号是3位的节拍编码t_2..t_0和8位的指令操作码IR15..IR8,能准确清楚地表明此行对应的是哪一条指令的哪一个节拍(操作步骤)。该行输出的是相关指令在这一个节拍中需要提供给运算器芯片的13位信号、提供给内存和串口芯片的3位控制信号。对这些信号的控制功能具体说明如下。

用于运算器的13位信号是aluoe,c0,I8..I6,I5..I3,I2..I0,A__,B__,其中:

① aluoe决定是否允许送出ALU运算结果到数据总线DB,低电平有效。当aluoe=0时允许送ALU的输出到DB;aluoe=1时ALU的输出为高阻态,会断开与DB的连接。

② c0是送给ALU最低一位的进位输入信号,可能为0或者1,要看指令这一步的操作功能。

③ I8..I6,I5..I3,I2..I0这3组3位信号的控制功能分别用于选择对ALU结果的处理方案、选择ALU的运算功能和ALU的运算数据,具体用法已经在本教材第4章的表4.1~表4.3中给出,这里不再赘述。

④ A__,B__表示在读写运算器中的16个寄存器时使用的2组4位的寄存器编码的方案,当[A__,B__]=[0,0]时,DR、SR的编码来自指令寄存器的IR7~IR4、IR3~IR0;当[A__,B__]=[0,1]时,DR、SR选择默认的寄存器R4(堆栈指针);当[A__,B__]=[1,0]时,DR、SR选择默认寄存器R0(仅用于IN、OUT指令)。

用于内存和串口的3位信号是MIO、REQ、WE,其中的MIO=0表明需要内存或串口工作,MIO=1表明不允许内存或串口工作;当MIO=0时,是内存还是串口工作决定于REQ,当REQ=0时是内存工作,REQ=1时是串口工作;当MIO=0时,是执行读操作还是写操作决定于WE,当WE=0时是写操作,WE=1时是读操作。3位信号的关系如表6.2所示。

表6.2 3位信号的关系

MIO	REQ	WE	运行功能
0	0	0	内存写
0	0	1	内存读
0	1	0	串口写
0	1	1	串口读
1	×	×	内存与串口都不运行

接下来对真值表的具体内容详细说明如下。

真值表的第1行对应指令的取指周期,所以该行的节拍编码为000,表明是取指周期,指令操作码是8位的不管位x,表明此时的操作功能与指令无关。

其余的4行对应4条指令的执行周期,所以这4行的节拍编码都为010,表明是指令的执行周期,指令操作码分别是ADD指令、AND指令、MVRD指令和JMPA指令的操作码。接下来可以用前面刚介绍过的内容,来分析、判断每一行中输出的16个信号实现的控制功能。

第1行执行到内存读取指令(MIO REQ WE=0 0 1)并保存到IR,还要完成PC+1的

增量功能(在控制器部件中完成),形成下一条指令的地址并保存到PC,此时不必使用运算器,有关的某些控制信号填写0值即可,但要确保运算器内部寄存器内容不被修改(I8～I6＝0 0 1),要禁止ALU的输出送DB(aluoe＝1),不影响用DB传送从内存读出的指令。

第2行和第3行是双寄存器之间的运算(I2～I0应为0 0 1),DR应接收运算结果(I8～I6应为0 1 1),运算功能分别是算术加(I5～I3应为0 0 0)和逻辑与(I5～I3应为1 0 0),无内存和串口读写操作(MIO REQ WE＝1 0 0),此时可以送ALU的输出到DB(aluoe＝0),以便通过DB的指示灯显示ALU的运算结果。

第4行和第5行都是到内存读取双字指令的第2个指令字(MIO REQ WE＝0 0 1),第4行中的JMPA指令读出的是指令转移地址,需要保存到PC(在控制器部件内部完成),此时不使用运算器,有关的某些控制信号填写0值即可,但要禁止送ALU的输出DB(aluoe＝1),运算器内部寄存器内容不被修改。第5行的MVRD指令从内存读出来的是一个立即数,需要保存到运算器的DR寄存器中,要通过把立即数D加上0送DR完成,因此I2～I0、I5～I3、I8～I6应为1 1 1、0 0 0和0 1 1。

这里要补充说明以下4个问题。

(1) 这个程序实现的4条指令是实现全部30条基本指令的ABEL程序的一部分,这样做是为了使这个程序尽可能地精简,方便学生学习和理解,更是为了给学生扩展新指令留下足够的选择空间。在此基础上,每扩展一条指令进去,只需在现有的ABEL程序中增加或修改很少数的语句(几行文字)即可完成。

(2) 若实现的不是全部30条指令,监控程序就运行不起来的,需要找出怎样向实验计算机送入调试程序的办法,可行办法之一是在MACH芯片内实现一个小ROM电路(仅16个字),用于编辑和保存学生设计的调试程序,这个小ROM电路很容易使用一张真值表进行描述,其输入信息是PC3～PC0,输出信息是16位的指令代码。此时要解决的问题是,指令不再是从内存芯片读出,而是由小ROM提供,这容易在ABEL程序中处理。小ROM中的程序只能使用指令代码,指令的运行步骤、控制信号的状态、运行结果只能通过查看指示灯来了解。

(3) 也可以考虑把时序电路的接收使能需求也直接表示在真值表中,就像微程序控制器中采用的办法,我们没有这样做,一是希望使真值表的输出信号的位数尽量少;二是要求学生能够初步体验到写信号逻辑方程的过程和具体办法,好在这并不太难,在真值表的注释部分又给出了每一条指令的每一个步骤的执行功能。

6.3 微程序控制器部件

微程序控制器是控制器中的一种常用类型。1951年,英国剑桥大学的M. V. Wilkes教授首次提出了微程序控制的概念,其后陆续有人加入了研究的行列并取得重大进展,但由于找不到保存微程序内容的器件而不能使用。随着半导体器件的出现和发展,有了速度较快的存储器介质,微程序控制器进入了实用阶段,于1964年4月,使用微程序控制器的IBM360计算机研制成功,其后在性能要求不是特别高的系列计算机系统中得到普遍应用。其缺点是运行速度较慢,难以在性能要求特别高的计算机系统中使用。它的基本运行原理,是用多条微指令“解释”每一条指令的功能。硬件组成中的核心线路是一个被称为控制存储

器的部件(使用 ROM 器件实现),用于保存由微指令(指令一个执行步骤用到的控制信号的集合)组成的微程序。在程序执行过程中,将按照指令及其执行步骤,依次从控制存储器中读出一条微指令,用微指令中的微命令字段控制各执行部件的运行功能,并用下地址字段形成下一条微指令的地址,使得微程序可以连续运行。

6.3.1 微程序控制器的基本组成和运行原理

微程序控制器的计算机硬件系统组成如图 6.10 所示,虚线框内的部分是控制器部件。

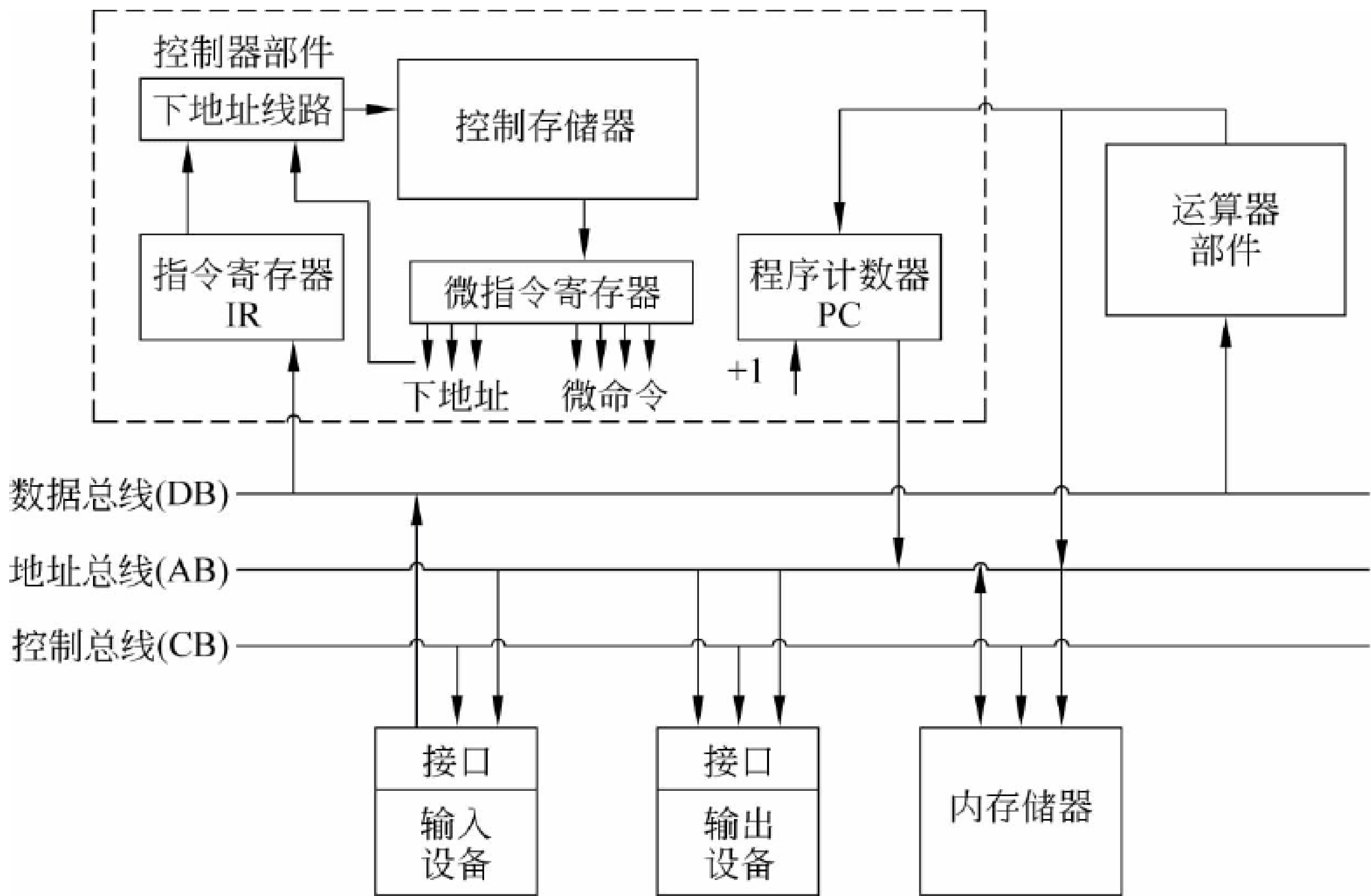


图 6.10 选用微程序控制器的计算机硬件系统组成

从图 6.15 中可以看到,微程序控制器由 4 个子部件组成。

(1) 第 1 个是程序计数器 PC,用于保存一条指令在内存中的地址,服务于读取指令,通常需要有增量功能,并可接收改变指令执行次序的指令地址。PC 的内容是下一条将要执行的指令在存储器中的单元地址,解决的是按照程序中规定的指令次序有序读出指令的问题。

(2) 第 2 个是指令寄存器 IR,用于保存从内存读来的指令内容,以便提供执行指令的过程中要用到的指令本身的主要信息。

(3) 第 3 个是控制存储器(简称控存)和微指令寄存器,控存用于保存微程序,微指令寄存器暂存从控存中读出的一条微指令,设置微指令寄存器可以提高微程序的执行速度。控制存储器提供类似于硬布线控制器中的控制信号产生部件的功能。

(4) 第 4 个是微指令下地址线路,用于向控制存储器提供读操作使用的地址,解决的是读出和执行微指令的次序问题。这个地址值与指令操作码、微指令字中下地址字段的内容等有关。下地址线路提供类似于硬布线控制器中的节拍发生器的功能。

要理解微程序控制器的运行原理,首先需要明确指令和微指令的关系与区别,从它们的内容组成和所实现功能两个方面来看,指令和微指令属于不同层面上的概念。

指令是指示计算机硬件系统完成一项最基本的运算或者操作功能的命令,使用的全部指令组成一台计算机的指令系统,用于设计完成各种计算任务或者信息管理等功能的程序,

运行中的程序将保存在主存储器中。指令是程序设计人员与计算机系统沟通和交互的媒介。

微指令则是直接控制计算机硬件线路完成指令功能的控制信号的集合,被划分为微命令字段和下地址字段两大部分。计算机厂家用微指令设计"解释"每一条指令执行过程的微程序,微程序被固化在控制存储器中。微指令是计算机指令和硬件电路之间建立联系的媒介,计算机的使用人员通常接触不到微程序和微指令的内容。

可以从以下几个角度来理解微程序控制器的基本组成和运行原理。

(1) 有关**程序计数器 PC 和指令寄存器 IR** 的功能已经在硬布线控制器的章节讲解过,在微程序控制器中没有变化,在此不再赘述。

(2) 在计算机系统中,执行每一条指令,都要经过读取指令、分析指令、执行这条指令规定的具体操作功能(这里可能还要再分为几个步骤)等操作步骤。在微程序控制器中,指令的每一个执行步骤被称为一个微周期(又称 CPU 周期),完成一项或几项运算或操作(此处称其为**微操作**)功能,要使用一条**微指令**提供控制计算机各部件完成这些微操作的控制信号(可能有几十位到一二百位),执行一条机器指令所使用的几条微指令组合成一段**微程序**。整个指令系统使用的全部微程序(几十、几百条或更多)被有序地整合成一个整体,并被固化在控制存储器中,这个控制存储器是微程序控制器的核心部件。在需要用到哪一条微指令的时候,要将其从控制存储器中读出,首先将其保存到微指令寄存器,之后再把微指令各字段中的控制信号传送到计算机的各功能部件。设置微指令寄存器是必要的,使执行本条微指令和读出下条微指令这两项操作的时间重叠起来,避免先用一段时间读出微指令,再用一段时间执行微指令,从而提高了微指令的执行速度。

(3) 在微程序控制器中,要通过微指令**下地址线路**提供下一条微指令在控存中的地址,表示和得到下条微指令地址有多种方案,更多内容将在 6.3.2 节讲解。从图 6.10 可以看到,下地址线路把指令的操作码和微指令中下地址字段的信息作为输入,输出的就是一条微指令的地址并送到控制存储器。微指令的内容被划分为下地址字段(提供形成下条微指令地址的有关信息,可能被细化为几个更小的字段,用于控制器部件本身)和微命令字段(由多个更小的字段给出多位的控制信号,用于控制计算机的各执行部件,如运算器、存储器、接口、总线等协同运行)。

以执行一条 ADD 加法指令为例,看如何通过微程序实现对这条指令执行过程的控制,如图 6.11 所示。先假定指令格式由操作码 OP 和 2 个寄存器(编号 B、A)组成,实现的功能为 B←(B)+(A),即寄存器 B 的内容与寄存器 A 的内容相加,再把运算结果保存回寄存器

ADD指令的执行步骤	读取指令		读寄存器堆	ALU加运算	结果写回
微指令的微命令字段的内容	指令地址送主存储器的微命令,读存储器的微命令,指令寄存器接收的微命令,实现PC←PC+1的微命令	无控制功能,但需要保证各寄存器的内容不被修改	读寄存器堆并缓存的微命令	ALU执行加法运算的微命令	结果写回寄存器堆的微命令
微指令的下地址字段的信息	顺序执行下一条微指令的标志信息	按指令操作码转到对应的微程序的标志信息	顺序执行下一条微指令的标志信息	顺序执行下一条微指令的标志信息	无条件转移到下一条指令的取指步骤的信息

图 6.11　ADD 指令的执行步骤和用到的几条微指令

B中。指令选用4个执行步骤的方案完成：①读取指令并完成PC增量；②读寄存器组中的两个数据并缓存；③ALU运算并缓存结果；④结果写回寄存器B。为此需要使用4条微指令提供这4个步骤用到的控制信号，这4条微指令都要在微命令字段和下地址字段给出两类不同用途的信息。在微命令字段给出的是控制计算机各部件完成ADD指令功能的控制信号，下地址字段给出的是指明微指令之间执行次序(彼此之间的连接方式)的有关信息。从图6.16中可以看到，在读取指令和读寄存器堆两个步骤之间插入了一条微指令，是运行微程序控制器所要求的。因为在取得指令之后，需要用指令操作码形成对应本条指令的微程序的首地址，并从控制存储器中读出这条微指令，这个读操作需要占用一个微周期才能完成，此期间计算机各执行部件处于空闲状态。因此这条指令实际上是用5个步骤完成的。

几条微指令之间的执行次序，是通过微指令地址的连接方式体现出来的。第1条微指令的地址是由上一条机器指令的最后一个执行步骤准备好的，完成的是两条机器指令之间的衔接关系。若在控制存储器中，把第2条微指令排在第1条微指令的后面，则只需在第1条微指令的下地址字段中指定顺序执行方式；同理，第3条微指令结束时进到第4条微指令，第4条微指令结束时进到第5条微指令都可以选用顺序执行方式实现。第5条微指令则要在自己的下地址字段指定无条件地转移到第1条微指令，以便进到下一条机器指令的取指步骤。但在处理第2条微指令进到第3条微指令的时候略显复杂，需要在第2条微指令的下地址字段指定按指令操作码实现功能分支，需要用到一个专用的线路把指令操作码映射成对应的微程序入口地址，用这个地址去读控制存储器，读出对应这条机器指令的微程序的首条微指令。综上所述，这里的5条微指令中的前2条公用于所有指令，是执行每一条机器指令必须经过的操作步骤；后面的3条可以依据指令不同而不相同。

6.3.2 微程序设计中的下地址形成逻辑和微程序设计

1. 得到下一条微指令地址的有关技术

要保证微指令逐条有序执行，就必须在本条微指令的执行过程中，可取来或临时形成(产生)下一条微指令的地址；从运行效率考虑，稍后还应能用此地址把下一条微指令的内容从控制存储器中读出来，以便在本条微指令结束后，能尽快地进入下一条微指令的执行过程。

决定下一条微指令地址(简称下地址)的因素较多，处理办法各不相同，具体包括以下几个方面。

(1) 微程序顺序执行时，下地址为本条微指令地址加1。

(2) 在微程序必定转向某一微地址时，可以在微指令字中的下地址字段中给出该地址值。

(3) 按微指令(上一条或本条)的某一执行结果的状态，选择顺序执行或转向某一地址，此时必须在微指令字中指明判断所依据的条件及转移地址。要判断的条件，可以是运算器的标志位状态，控制器的执行状态，如多次的微指令循环是否结束，外设是否请求中断等。

(4) 微子程序的调用及返回控制，会用到微堆栈。

(5) 依条件判断转向多条微指令地址中的某一地址的控制，它可以是前述第(3)条的更复杂一点的用法。

(6) 依据取来的机器指令的操作码，找到对应该条指令的执行过程的一段微程序的入

口地址。这种情况,通常被称为微程序控制中的功能分支转移。

从上述讨论中可以看出,要得到下一条微指令的地址,至少得从以下两个方面入手。

(1) 要在微指令的下地址字段中,分配相应的几个字段,用于给出微指令转移地址(完整的一个地址,或部分的多个地址),以及是顺序执行、无条件转移或条件转移及其判断条件,是否是功能转移,是否是微子程序调用或返回等。

(2) 应有专门的硬件线路支持,用于实现诸如微指令地址加1,依据判断条件给出判定结果为真还是为假,给出微堆栈组织并实现入/出微堆栈管理,解决指令操作码与相关微程序段的对应关系,以实现微程序中的功能分支。例如,Am2910芯片就是用于形成下一条微指令地址的非常典型的核心硬件。

2. Am2910 芯片的功能和内部组成

Am2910是一片能提供12位微指令地址的器件,即它的输入输出的地址位数和器件内的部件位数均为12位,能直接寻址4096条微指令字的空间范围,其内部结构如图6.12所示。

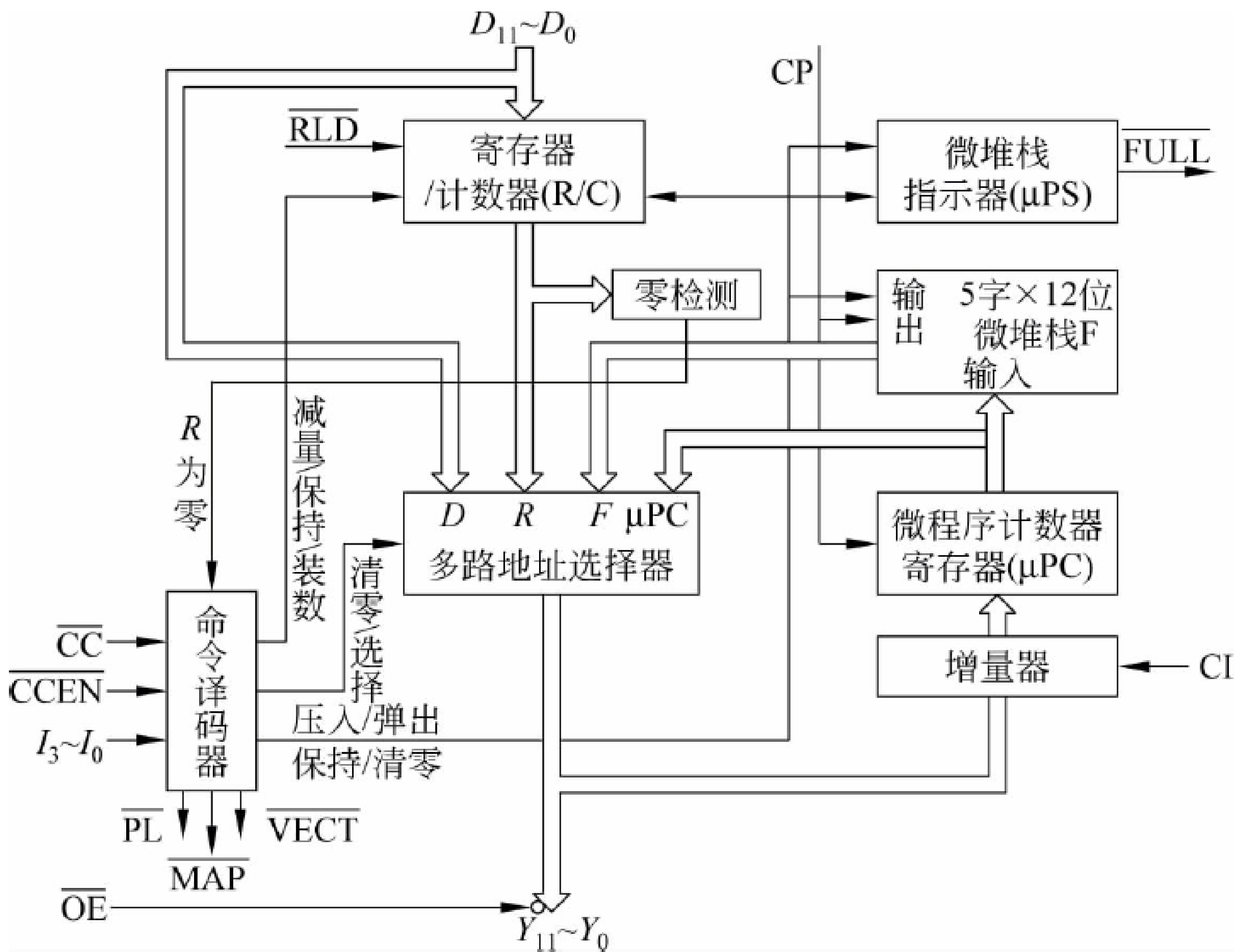


图 6.12 Am2910 内部结构框图

1) Am2910 的内部组成

(1) Am2910包括一个4输入的多路地址选择器,用来从寄存器/计数器(R/C)、直接输入(D)、微程序计数器(μPC)或微堆栈(F)这4个输入中,选择其一作为下一条微指令的地址。

(2) 寄存器/计数器由12个D型触发器组成。当它用作寄存器时,主要用于保存一个微地址,用以实现微程序转移;当它用作计数器时,具有减1功能(何时减1,取决于Am2910的命令码),主要用于控制微程序的循环次数,若装入的初值为N,则可能执行$N+1$次循环。

(3) 微程序计数器由12位的增量器和12位的寄存器μPC组成。当增量器的进位输入CI为高电平时,多路器的输出Y加1后装入μPC(即μPC←Y+1),用于实现微程序的顺序

执行;而当CI为低电平时,多路器的输出Y直接装入μPC(即$\mu PC \leftarrow Y$),用于实现同一条微指令的多次执行。

(4) 微堆栈是由5字×12位的寄存器堆栈和微堆栈指针μSP组成,主要用于保存微子程序调用时的返回地址和微程序循环的首地址。微堆栈指针μSP总是指向最后一次压入的数据,因此,执行微程序循环时,允许不执行弹出操作而直接访问微堆栈的栈顶。当微堆栈中的数据达到5个时,就发出微堆栈已满信号($\overline{FULL}=0$),这时,任何压入操作都将覆盖掉栈顶的原有数据。

(5) 命令译码器接收外部送来的命令码$I_3 \sim I_0$,条件输入$\overline{CC}$和条件允许$\overline{CCEN}$信号,并对其译码,产生芯片内工作需要的控制信号和外部要用的3个控制选择信号$\overline{PL}$、$\overline{MAP}$和$\overline{VECT}$。3个输出信号$\overline{PL}$、$\overline{MAP}$和$\overline{VECT}$用于决定外部直接输入D的来源。

当$\overline{PL}$有效时(即$\overline{PL}=0$),D来源于微指令的下地址字段,用于给出微程序转移地址。

当$\overline{MAP}$有效时(即$\overline{MAP}=0$),D来源于MAPROM,用于实现从机器指令的操作码找到相应的微程序段首地址;当$\overline{VECT}$有效时(即$\overline{VECT}=0$),原意是来源于向量,现直接接地,未使用。

2) Am2910引脚的定义

(1) 输入信号

① $D_{11} \sim D_0$:外部直接输入的数据,既可作为寄存器/计数器的初值,也可以经过地址多路选择器直接从Y输出,作为下一条微指令的地址。

② $I_3 \sim I_0$:Am2910的命令码,来自微指令字相关字段,用以选择Am2910的16条命令。

③ $\overline{CCEN}$和$\overline{CC}$:共同确定测试条件是否通过,若$\overline{CCEN}$为低且$\overline{CC}$为高,则指明测试失效;而$\overline{CCEN}$为高或$\overline{CC}$为低,均表明测试通过。若把$\overline{CCEN}$接地,即使其恒为低电平,则可以只使用$\overline{CC}$判断测试结果,$\overline{CC}$为低是测试通过,$\overline{CC}$为高则表明测试失效,我们在实验计算机中就是这样用的。此时它与命令码$I_3 \sim I_0$共同决定给出下一条微指令地址的方案和对堆栈的操作。

④ $\overline{RLD}$:寄存器/计数器装入信号,当其为低电平时,不管Am2910所执行的命令和测试条件如何,都强制把直接输入$D_{11} \sim D_0$装入Am2910内部的寄存器/计数器。

⑤ CI:增量器进位输入,当其为高电平时,控制微指令地址增量,即执行$\mu PC \leftarrow Y+1$,当其为低电平时,执行$\mu PC \leftarrow Y$。

⑥ $\overline{OE}$:Y输出允许信号,低电平有效,当为高电平时,Y输出为高阻态。

⑦ CP:时钟脉冲信号,由低变高的上升边沿触发所有内部状态的变化。

(2) 输出信号

① $Y_{11} \sim Y_0$:下一条微指令的地址,它直接被用作读控制存储器的地址。

② $\overline{FULL}$:微堆栈满信号,低电平有效。

③ $\overline{PL}$,$\overline{MAP}$,$\overline{VECT}$:3个使能信号,用于决定直接输入D的3个来源,加上芯片内的多路地址选择器的另外3个输入R、F、μFC,$Y_{11} \sim Y_0$是经过6选1得到的。

3) Am2910芯片的功能与具体用法

表6.3给出了Am2910所完成的部分功能,这些功能由命令码$I_3 \sim I_0$、条件输入$\overline{CC}$和

$\overline{CCEN}$以及计数器当前值的组合结果来决定。

Am2910 提供了 16 条命令，用来控制 Am2910 内部的操作和选择下一条将要执行的微指令的地址。其中只有少数命令（如 0，2 和 14 号命令）的执行结果仅由命令码本身决定，大部分命令还都要受到测试条件（$\overline{CC}$和$\overline{CCEN}$）为真还是为假的控制，有些命令（如 8，9 和 15 号命令）的执行结果，则要受到内部计数器当前值是否为零的控制，其中 15 号命令同时受到内部计数器的值是否为零和测试条件是否通过的双重控制，以实现 3 路微程序转移的功能。

现将实验计算机设计中要用到的 0，2，3，14 这 4 条命令的功能和我们的具体用法说明如下。

（1）**0 号命令**。用于初始化，即无条件清除内部微堆栈，并使 Y 的输出一定为零，用于系统加电时，确保此时系统从 0 号微地址开始执行微程序。

（2）**2 号命令**。用于指令功能分支，即输出信号$\overline{MAP}$为低，使 D 输入信号从 MAPROM（微地址映射部件）得到，并将其作为输出微地址 Y 的值，实现用指令操作码找到对应该指令的微程序段的入口地址，从而开始该条指令的执行过程。

（3）**3 号指令**。用于条件微转移控制，当条件成立，即$\overline{CC}$为低时，用$\overline{PL}$把微指令字中的下地址字段的内容（转移地址）经过 D 输入并送到 Y，实现微程序转移。当$\overline{CC}$为高时，微程序顺序执行，即把已增 1 后的微指令地址作为下地址。若外部电路确保送入的$\overline{CC}$的状态为低，3 号条件微转移命令也可以用于实现无条件的微程序转移来使用。

（4）**14 号命令**。顺序执行，即执行紧跟在本条微指令后面的那条微指令。

表 6.3　Am2910 器件的功能控制

指令号	完成功能	R/C 内容	R/C 操作	使能信号	$\overline{CCEN}$=0			
					$\overline{CC}$高		$\overline{CC}$低	
					Y 输出	堆栈	Y 输出	堆栈
0	初始化		/	$\overline{PL}$	0	清除	0	清除
2	指令功能分支		/	$\overline{MAP}$	D	/	D	/
3	条件转移		/	$\overline{PL}$	μPC	/	D	/
14	顺序执行		/	$\overline{PL}$	μPC	/	μPC	/

说明：①若测试失败则保持，否则就装数；μPC——微指令计数器；D——直接输入；R/C——寄存器/计数器。②图中符号“/”表示保持原内容不变。

6.3.3　TEC-XP-Ⅱ教学计算机的微程序控制器的设计与实现

设计一台计算机的首要任务是确定设计目标和市场定位，包括系统的性能与经济指标，指令格式和指令系统，总体技术方案等。

其次是初步设计系统的硬件构成，设计并通过仿真测试指令系统；细化系统硬件结构，划分指令执行步骤并确定每一步骤的功能；确定计算机各部件需要使用的控制信号，设计微指令格式，包括下地址字段和微命令字段的内容组成。

接下来是设计微程序的内容，其中微命令字段的划分和实现的控制功能非常类似于硬布线控制器的设计结果，此处不再赘述。微指令字中下地址字段的内容选择和使用方法是微程序控制器设计的特殊内容。

最后一步是把设计的微程序中的每一条微指令都安排到控制存储器的一个存储单元中，这一设计步骤可能要反复几次，在微指令的下地址字段的内容和这条微指令在控制存储器中的地址之间建立正确的对应关系，包括指令的操作码与对应这条指令具体操作功能的微程序段的入口地址的对应关系。

设计完之后，将进入调试修改的阶段，直到得到满意的设计与运行结果。

在有了能力很强的硬件描述语言的软件工具和高集成度的可编程芯片之后，设计实现硬布线的控制器或者微程序的控制器都不再那么困难。

教学计算机系统实现了硬布线和微程序两种方案的控制器，是采用两个独立的 ABEL 程序分别设计的。在设计的整个过程中，特别强调保持二者之间尽可能多的一致性。例如力争做到使微指令字中的微命令字段的构成及具体内容与硬布线控制器的控制信号尽量保持一致，则在实现了硬布线控制器之后，其设计结果的很大一部分内容可以较为简单地复制到微程序的控制器中。下面给出这个微程序控制器的设计结果，在图 6.13 给出了微程序的结构和微指令的执行流程，之后给出描述微程序控制器组成及其功能的 ABEL 程序的部分内容，程序中加了比较多的注释，在程序之后还要对其中的关键语句进行简要说明。

教学计算机实现的是在本教材的 5.3.4 节讲解过的实验计算机的 30 条基本指令，被划分为 A、B、D 这 3 组，分别用 2 个、3 个、4 个步骤完成，这几个步骤分别称为取指步骤、执行步骤、存储器读写步骤_1 和存储器读写步骤_2，已经在图 6.13 中表示清楚。在微程序控制器中，指令的每一个步骤要用到一条微指令，取指操作公用于所有指令，使用一条微指令；取指之后，指令进入具体功能的执行阶段，21 条 A 组指令在执行步骤通常应各用一条微指令，但由于 5 条相对转移指令可以合用一条微指令，故 A 组指令共使用了 17 条微指令；8 条 B 组指令各使用 2 条微指令，先在执行步骤准备内存地址，后到存储器读写步骤_1 完成读写操作，合计使用 16 条微指令；CALA 指令用 3 个步骤完成，要用 3 条微指令，在执行步骤取子程序入口地址到 PC 并暂存主程序断点，在存储器读写步骤_1 修改堆栈指针并送 AR，在存储器读写步骤_2 写主程序断点进堆栈。

有 3 件事情需要再次澄清。

(1) 在微程序控制器中，取指之后需要用一个步骤完成指令的功能分支操作，就是使用由指令操作码映射出的微指令地址去读控制存储器，以便得到对应刚读出的那条指令要用到的微指令，此时计算机各执行部件由于尚无得到需要的控制信号而处于空闲状态。

(2) 需要正确给出微程序的首地址，可以在系统复位(按下设备的 RESET 按键)时，把取值为 0000 的命令码 CI_3～CI_0 送到 Am2910 芯片，则芯片会送出 Y_7～Y_0 为全 0 的控存地址，使微程序从 0♯ 微指令开始运行，之后微指令的下地址电路就会自行提供出下条微指令地址，确保微程序得以连续运行。

(3) 在系统复位时还把 PC 内容清 0，使系统启动后将从内存的 0 地址启动监控程序。

确定微指令的格式很重要，我们的方案是使用 34 位字长的微指令，分为 3 个大字段。

(1) 微指令的下地址信息字段，占用 11 位（可以变化），包括 6 位的微指令的转移地址 nadr5～nadr0，Am2910 的 4 位命令码 CI3～CI0，1 位的微指令是否转移的条件码 _CC。

(2) 时序电路的输入控制字段，占用 7 位，时序电路包括 flag_c、flag_z、pc、ar 的接收控制信号，包括 flg_c2～flg_c0，pc_c1～pc_c0，ar_c 共 6 位，另一位 ar_AB 信号用于选择送地址总线 AB 的信息来源。

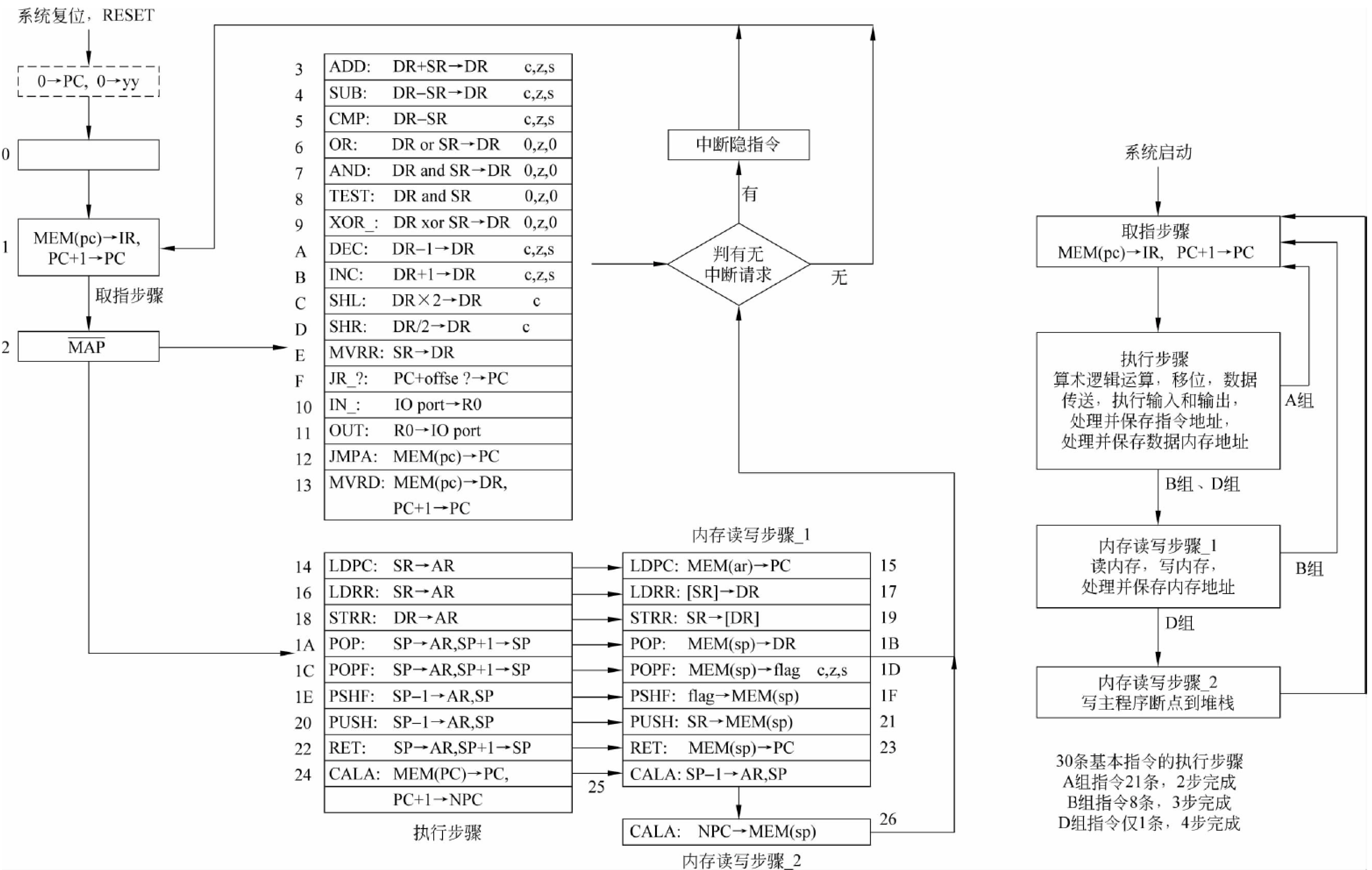


图 6.13 实现 30 条基本指令的微程序结构和微指令执行流程

(3) 微指令字的微命令字段,占用 16 位,与硬布线控制器使用的 16 位的控制信号完全相同,因此可以把硬布线控制器的设计结果简单地复制到微程序控制器的相应部分,使微程序控制器的设计主要集中到确定微指令字的下地址字段内容。设计完微指令格式后,就可以把全部微指令安排到控制存储器中,从图 6.13 可以看到设计的结果。

描述实现 4 条典型指令的 ABEL 程序清单如下。

```
MODULE TEC_new
TITLE  'controller component'
                                    "a_16_mc_1_new_160913_bak_1.abl
DECLARATIONS
RESET,CLK       pin 151,68;                             "系统复位和时钟
IR15..IR0       pin 64..57,54..47;                      "IR送来的指令字
IR_G,IR_clk     pin 174,175;                            "IR接收控制 (低电平有效)
Cy, Zero        pin 169,171;                            "Am2901产生的标志位信息
AB15..AB0       pin 87..80,77..70;                      "地址总线
DB15..DB0       pin 24,23,26,25,28,27,30,29,32..39;     "数据总线
MIO, REQ,WE     pin 95,94,93  istype 'dc,com';          "控制内存和串口的信号
I8..I0          pin 14..21, 135 istype 'dc,com';        "控制运算器的信号
B3..B0,A3..A0   pin 9..12, 5..8 istype 'com';
aluoe,ram15,ram0,c0     pin 136,137,139,141 istype 'com';
F_C,F_Z                 pin 168,170;                    "显示标志位
pc15..pc0,ar15..ar0     node istype 'reg,keep';         "程序计数器、地址寄存器
npc15..npc0             node istype 'reg,keep';         "暂存中断断点的寄存器
flag_c,flag_z           node istype 'reg,keep';         "标志位触发器
A_,B_ ,jr_5, DB_oe      node istype 'com';              "中间信号、允许 MACH送信息到 DB
sum15..sum0,jr_zu       node istype 'com';              "专用加法器电路,相对转移指令组
wk15..wk0,cy15..cy1     node istype 'com';              "加法器一路输入、每位进位输出
flg_c2..flg_c0, pc_c1,  pc_c0,pc_oe,                    "flag和 pc的接收控制
     ar_oe, ar_AB       node istype 'com';              "ar的接收控制,ar的输出送 AB
"--------连接 Am2910芯片的 MACH  IO管脚和微程序控制器组成 ---------------
CI3..CI0,_CC,CCEN  pin 106..103,100,102;                "Am2910的 4位命令码,转移控制
YY5..YY0           pin 124,125,122,123,120,121;         "Am2910送来的微指令地址
DD7..DD0           pin 118,115,116,113,114,111,112,109 ;   "送下条微指令地址信息到 Am2910
_MAP,_PL           pin 98,96;                           "Am2910送来的 2个控制信号
m_ir33..m_ir0      node istype 'reg,keep';              "微指令寄存器
y_y5.. y_y0        node istype 'reg,keep';              "当前微指令地址
sig33 .. sig0      node istype 'com';                   "控制存储器
madr5.. madr0      node istype 'com';                   "映射 IR_op为微指令首地址
con3..con0         node istype 'com';
c,z,x= .C.,.Z.,.X.;                                     "说明常量和集合
flg_c= [flg_c2..flg_c0];  pc_c= [pc_c1,pc_c0]; "Flag、 PC的接收使能信号
m_ir= [m_ir33..m_ir0] ;yy= [y_y5..y_y0];
pc= [pc15..pc0];IR= [IR15..IR0]; ar= [ar15..ar0];
DB= [DB15..DB0];AB= [AB15..AB0]; npc= [npc15..npc0];sum= [sum15..sum0];
ir_op= [IR15..IR8];                                     "4条指令的操作码和汇编语句名
```

```
ADD = ir_op= = ^h00;   AND = ir_op= = ^h02;
MVRD= ir_op= = ^h88;   JMPA= ir_op= = ^h80;
EQUATIONS
  jr_zu= (yy= = ^h0f);                          "用于专用加法器 ADDER的信号
  [wk15..wk1]= jr_zu&[IR7,IR7,IR7,IR7,IR7,IR7,IR7,IR7,IR7..IR1];
  wk0= jr_zu&IR0# !jr_zu;    [con3..con0]= [0,0,0,0];
"  jr_5= JR # JRNC&!flag_c# JRC&flag_c# JRNZ&!flag_z# JRZ&flag_z;   转移条件为真
                                      "微指令的各字段信号到 MACH输出引脚
  [DD5..DD0]= !_MAP&[madr5..madr0]# !_PL&[m_ir33..m_ir28];        "11位下地址字段信息
  [CI3..CI0]= RESET&[con3..con0]# !RESET&[m_ir27..m_ir24];        "Am2910的命令码
  _CC=m_ir23;     CCEN= 0;                                        "微指令转移条件
  [aluoe,c0, I8..I0, A_,B_, MIO,REQ,WE]= [m_ir22..m_ir7];         "微指令的微命令字段
!IR_G= (yy= = ^h00);   IR_clk= CLK;                              "IR的接收条件和时钟信号
  F_C= flag_c;          F_Z= flag_z;                              "显示标志位信息
[pc, ar, flag_c, flag_z, npc, m_ir, yy].clk = CLK;                "指定寄存器时钟信号
pc.oe= pc_oe;pc_oe= RESET# !RESET&(pc_c= = [0,1])# (pc_c= = [1,0]);
when RESET then {pc:= 0;yy:= 0;} else                             "设置监控程序和微程序的起始地址
{[m_ir33..m_ir0]:= [sig33..sig0];   yy:= [YY5..YY0];              "微指令寄存器、yy接收
  when (pc_c= = [0,1]) then pc:= sum;                             "PC+ 1/offset→PC
  when (pc_c= = [1,0]) then pc:= DB;                              "双字指令的第 2个指令字→PC
}
  flag_c= (flg_c= = [0,0,1])& Cy  # (flg_c= = [0,1,0])& 0         "flag_c接收
# (flg_c= = [1,0,1])& ram0# (flg_c= = [0,0,0])& flag_c;
  flag_z= (flg_c= = [0,0,1])& Zero # (flg_c= = [0,1,0])& Zero     "flag_z接收
# (flg_c= = [0,0,0])& flag_c;
  when ar_oe then ar:= DB;   else ar:= ar;                        "AR接收内存地址
  when ar_AB then AB= ar;   else AB= pc;                          "选择地址总线信息

when ([A_,B_]= = [0,0]) then {[B3..B0]= [IR7..IR4]; [A3..A0]= [IR3..IR0];}
when ([A_,B_]= = [0,1]) then {[B3..B0]= [0,1, 0,0]; [A3..A0]= [0,1, 0,0];}
when ([A_,B_]= = [1,0]) then {[B3..B0]= [0,0, 0,0]; [A3..A0]= [0,0, 0,0];}
  ram15= 0;        ram0 = 0;                      "逻辑右移、左移指令的移位输入信号
  ram15.oe= !I7;   ram0.oe = I7;                  "移位管脚输入输出的三态控制

@ include 'adder_pc_mc_1.abl'                     "描述指令地址加法器的程序段
TRUTH_TABLE                                       "映射 IR_op 为微指令地址--MAPROM电路
   ([IR15..IR8]   -> [madr5..madr0])
[0,0,0,0,0,0,0,0]-> [0,0,0,0,1,1];   [0,0,0,0,0,1,0,1]-> [0,0,1,0,0,0]; "ADD   AND
[1,0,0,0,1,0,0,0]-> [0,1,0,0,1,1];   [1,0,0,0,0,0,0,0]-> [0,1,0,0,1,0]; "MVRD   JMPA

                          "微程序清单--控制存储器电路
TRUTH_TABLE" ([nadr5..nadr0,CI3..CI0,_CC,aluoe,c0,  I8..I0, A_,B_,MIO,REQ,WE,
" flg_c2..flg_c0,pc_c1,pc_c0,ar_oe,ar_AB])
([YY5..YY0] -> [sig33..sig0])
[0,0, 0,0,0,0]-> [0,0,0,0,0,1, 0,0,1,1, 0,  1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 1,0,0, 0,0,0, 0,0, 0,0];
```

```
[0,0, 0,0,0,1]->[0,0,0,0,1,0, 0,0,1,1, 0,  1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 0,0,1, 0,0,0, 0,1, 0,0];
                                                                          "MEM(pc)→IR, PC+1→PC
[0,0, 0,0,1,0]->[0,0,0,0,0,0, 0,0,1,0, 0,  0,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 1,0,0, 0,0,0, 0,0, 0,0];
                                                                          "MAP执行功能分支
[0,0, 0,0,1,1]->[0,0,0,0,0,1, 0,0,1,1, 0,  0,0, 0,1,1, 0,0,0, 0,0,1, 0,0, 1,0,0, 0,0,1, 0,0, 0,0];
                                                                          "ADD  DR+SR→DR   c,z
[0,0, 0,1,1,1]->[0,0,0,0,0,1, 0,0,1,1, 0,  0,0, 0,1,1, 1,0,0, 0,0,1, 0,0, 1,0,0, 0,1,0, 0,0, 0,0];
                                                                          "AND  DR&SR→DR   0,z
[0,1, 0,0,1,0]->[0,0,0,0,0,1, 0,0,1,1, 0,  1,0, 0,0,1, 0,0,0, 0,0,0, 0,0, 0,0,1, 0,0,0, 1,0, 0,0];
                                                                          "JMPA MEM(pc)→PC
[0,1, 0,0,1,1]->[0,0,0,0,0,0, 0,0,1,1, 0,  1,0, 0,1,1, 0,0,0, 1,1,1, 0,0, 0,0,1, 0,0,0, 0,1, 0,0];
                                                                     "MVRD MEM(pc)→DR,  PC+1→PC
END
```

这个程序是从描述30条基本指令的ABEL程序中摘录出来的,不够完整,但作为例子还是合适的,可以更简明一些。

程序中的说明段的内容,大部分与硬布线控制器中的说明相同,多出了与Am2910芯片相连接的管脚说明,多出了微程序控制器的专用电路,包括控制存储器、微指令寄存器、当前微指令地址、映射指令操作码和微程序段入口地址的电路等。说明部分给出的是设计中的规定,可增减但不能轻易变更设计,例如器件管脚号、指令和微指令格式、指令编码、微指令字段安排等不能修改,变更造成的错误会使系统不能运行,还可能损坏器件。

程序中的逻辑描述段,以@include 'adder_pc_mc_1.abl' 的方式把一段ABEL程序引入到本程序中,它实现的功能是pc+1或者pc+offset(带进位扩展)→pc,专用于计算指令地址,用到的只是线路设计知识,没有必要把这段程序直接写在本程序中。

这个程序的核心部分是一些逻辑方程语句和两个真值表。

(1) 逻辑方程语句主要用于描述触发器或者寄存器电路的接收功能,包括程序计数器PC、内存地址寄存器AR、保存ALU进位的触发器flag_c、结果为0的触发器flag_z,这些时序电路可以通过.ce的方式指定它们有接收使能控制,例如pc.ce=pc_ce;pc_ce=RESET # !RESET&(pc_c==[0,1])#(pc_c==[1,0]);**仅在pc.ce=1时,pc才能够接收输入,否则pc内容将保持不变**,哪些情况下pc需要接收输入已经用pc_c1和pc_c0两位信号写在对应的微指令字中。接收的内容采用条件赋值语句另行给出。其他几个时序电路也是类似处理。在硬布线控制器中,我们就没用选用这种办法,而是由设计者自己来设计这些使能信号的逻辑方程。

微指令寄存器m_ir和当前微指令地址寄存器yy在每一个时钟周期都要执行接收操作,就不能通过.oe对其说明。

这里需要强调一个重要概念,当前正在执行的微指令的内容是在此前的一个步骤从控制存储器读出来的sig33~sig0,读控存使用的地址是那一时刻由Am2910提供的YY5~YY0,需要在前一步的结束时刻把这两部分内容分别接收到m_ir和yy寄存器,到了本步骤,才能够用m_ir的输出控制各部件运行,并用yy的输出表明当前时刻并被使用在逻辑方程语句中。当前yy的内容是前一个步骤的YY5~YY0的值,当前的YY5~YY0的内容是读出下一条微指令的控存地址。

(2) 真值表主要用于描述控制存储器存储的信息，即每一条微指令的内容。其输入信号是控存的地址信息 YY5～YY0(由 Am2910 芯片提供)，输出信号是一条微指令字的具体内容 sig33～sig0，包括 11 位的下地址字段信息、7 位的时序电路接收输入的控制信息、16 位的微命令字段信息。

m_ir 中的 16 位微命令字段的内容要送到 MACH 芯片的输出管脚，用于控制运算器、内存和串口，具体实现的控制功能已经在硬布线控制器的章节讲解过了，这里不再赘述。

11 位的下地址字段信息用于确定下一条微指令的地址，对需要顺序执行的微指令，应使命令码 CI3～CI0＝1110，转移地址每位都填 0 即可。对需要转移的微指令，应使 CI3～CI0＝0011 且 _CC＝0，并在转移地址字段直接给出转移地址。还可以看到，微指令执行转移的情况比较多，顺序执行相对较少。在本设计中，微指令的转移地址几乎都为 00001，实现的功能是在指令结束后转到下条指令的取指步骤，尽量不再使用其他的微指令转移方式，这会使几条指令本来可以合用的一条微指令字重复几次出现在微程序中，好在微指令总条数并不多，多出少量微指令不会对系统产生影响。

7 位的时序电路的输入接收控制信息用于指出在哪些微指令的执行时刻，哪些时序电路需要执行接收输入，接收的信息是什么，这在多个不同时刻需要接收不同的输入内容的情况下，能够简化设计逻辑方程的工作，无须设计者自己去设计寄存器的接收控制信号的逻辑方程，又能避免误把真值表中的 YY5～YY0 用作当前微指令的执行时刻。我们在微程序清单(真值表)中，特意增加了每一条微指令对应哪一条指令的哪一个执行步骤、完成的基本功能是什么、标志位寄存器是否需要变动等内容。这些信息在微程序控制器的设计过程中是有用的，对学生看懂、理解微程序控制器的运行机制也会有所帮助。

另外一张真值表用于描述 MAPROM 电路，实现的功能是映射指令操作码为微指令地址，其输入信号是 8 位的指令操作码，输出信号是 6 位的微指令地址 madr。选用了真值表，设计者就无须自己设计 madr 每一位的逻辑方程，而交由 ABEL 的编译软件完成。

本章内容小结和学习方法建议

本章重点讲授简单计算机控制器的功能、组成与实现，这对于本课程来说是比较难学、但又是很重要的部分。教学安排以指令的执行过程(步骤)为主线索，把通用原理性知识与真实计算机控制器实例相结合，以硬布线方案的控制器为主，并简单介绍微程序控制器的基础知识。

学过本章之后，学生应该重点掌握的内容是控制器的功能与组成和运行控制。硬布线控制器的功能是按照指令及其所处的执行步骤向计算机各个功能部件提供它们协调运行所需要的控制信号，在计算机硬件系统中发挥指挥控制的作用。其基本组成包括程序计数器 PC(提供指令地址)、指令寄存器 IR(保存指令内容)、节拍发生器 Timing(指出指令的执行步骤)、控制信号产生部件 CU(用于向计算机各部件提供每条指令、每个执行步骤的控制信号)这 4 个部分。对微程序控制器，更希望通过对比它和硬布线控制器在功能、组成、所用电路、运行速度、适用场合等几个方面的相同与差异之处的方式来学习。

从指令执行步骤区分 3 种方案。有单指令周期的方案，所有指令都安排在同一个时钟周期完成，造成资源利用率和系统运行效率都低，实用性差；多指令周期的方案，不同指令选用不

同的执行步骤完成,资源利用率和系统运行效率居中,曾得到广泛应用,目前使用较少,但作为计算机组成原理这门课程,仍属于重点教学内容,也是学习指令流水线方案的基础;指令流水线的方案,资源利用率和系统运行效率最高,目前被普遍应用,多数人更倾向于把它划归到计算机系统结构课程的教学范围之内,在本教材中把这部分内容单独安排到第 13 章中进行讲解。后两种方案中,每条指令都是经过读取指令、指令译码和指令执行(可能又细分为几个具体步骤)这几个步骤完成的,指令的执行步骤是明确表现出来的,而在单指令周期方案中,每条指令的执行过程同样要按照读取指令、指令译码和指令执行的时间序列依次启动并开始有效运行,只是开始后就要一直持续下去直到读出下一条指令,而不是分阶段依次完成。

学习控制器,需要通过具体的实际例子(某种特定情况)来加深理解,既不能只是很抽象空泛地知晓某些原理知识,又不能被特例的某些实现细节所迷惑,处理好二者的关系十分重要。在教材中给出一个加了详细注释的、描述简单 CPU 系统的组成和功能的 ABEL-HDL 语言的源程序,并能够在配套的教学实验设备上调试运行,查看运行结果,在此基础上又可以对其加以修改和完善,开展学生自己的设计与实现工作,成为一个非常好的教学过程和实践环境。我们希望让学生真实感受到看懂这个程序确实不是太困难的事情,开展自己的设计与实现工作也是办得到的。如果教材中不提供一个可行、简单的 CPU“蓝本”,讲完基本原理就要求学生开展设计实现工作,其难度就太大了一些。若能够用 VHDL 语言设计一个简单的 CPU,又能围绕 FPGA 芯片实现一个基本的整机系统是非常理想的学习过程,可以取得非常好的教学效果,但难度要更大一些,在与本教材配套的实验教程中给出了一个称之为 FPGA-CPU 的例子,有兴趣与有精力的同学不妨看看此程序。

习题与思考题

1. 控制器的功能是________。

A. 向计算机各部件提供控制信号　　B. 执行语言翻译

C. 支持汇编程序　　D. 完成数据运算

2. 从资源利用率和性能价格比考虑,指令流水线方案________,多指令周期方案________,单指令周期方案________。

A. 最好　　B. 次之　　C. 最不可取　　D. 都差不多

3. RISC 比 CISC 使用的指令条数________,只用硬件实现那些功能________的指令,指令每个执行步骤用时________。

A. 更多　　B. 更少　　C. 更简单　　D. 更强大

E. 更长　　F. 更短　　G. 相当接近

4. 指出下面各题的对错,并简单说明理由。

(1) 程序计数器 PC 主要用于解决指令的执行次序。

(2) 程序计数器 PC 可以提供数据在内存储器中的地址。

(3) 指令寄存器用于保存指令内容。

(4) 指令寄存器用于保存指令地址。

(5) 微程序控制器的运行速度一般要比硬布线控制器更快。

(6) RISC 结构的计算机在同等性能的情况下处理器芯片的设计和制作成本更低。

（7）在微程序控制器中不设置微指令寄存器也是可行的。

5. 简述计算机的控制器的功能和基本组成。微程序的控制器和硬布线的控制器在组成和运行原理方面的相同、不同之处表现在哪些方面？

6. 在硬布线控制器中，节拍发生器的功能是什么？节拍发生器属于什么类型的数字电路？

7. 硬布线控制器的时序控制信号产生部件是用什么类型的数字电路实现的？该部件的输入信号可能包括哪一些？

8. 控制器的设计和该计算机的指令系统是什么关系？

9. 微指令的下一条微指令地址通常有哪些来源？从微程序设计的角度，指出可能需要支持哪些微程序执行的流程结构？为此又应有什么样的硬件线路支持？

10. 在微程序的控制器的组成中，为什么通常总要设置微指令寄存器部件呢？

11. 指令采用顺序方式、流水线方式执行的主要差别是什么？各有什么优点和缺点？

12. 在指令流水线中，每一条指令本身的执行时间减少了吗？如果没减少，那么为什么还要采用流水线技术呢？一般来说流水线有哪些特点？总结流水线各种分类方法的原则。

13. 假设一条指令的执行过程分为“取指令”、“分析”和“执行”3段，每段的时间分别是Δt、$2\Delta t$和$3\Delta t$。在下列各种情况下，分别写出连续执行n条指令所需要的时间表达式。

（1）顺序执行。

（2）仅“取指令”和“执行”重叠。

（3）“取指令”、“分析”和“执行”重叠。

14. 什么是流水线的相关问题？通常都有哪几类相关问题？这些相关问题都是什么原因造成的？各种相关问题都有哪些解决方法？

15. 冯·诺依曼计算机中指令和数据均以二进制形式存放在存储器中，CPU区别它们的依据是________。

A. 指令操作码的译码结果　　B. 指令和数据的寻址方式

C. 指令周期的不同阶段　　D. 指令和数据所在的存储单元

16. 相对于微程序控制器，硬布线控制器的特点是________。

A. 指令执行速度慢，指令功能的修改和扩展容易

B. 指令执行速度慢，指令功能的修改和扩展难

C. 指令执行速度快，指令功能的修改和扩展容易

D. 指令执行速度快，指令功能的修改和扩展难

17. 下列选项中，能缩短程序执行时间的措施是________。

Ⅰ. 提高CPU时钟频率　Ⅱ. 优化数据通路结构　Ⅲ. 对程序进行编译优化

A. 仅Ⅰ和Ⅱ　B. 仅Ⅰ和Ⅲ　C. 仅Ⅱ和Ⅲ　D. Ⅰ、Ⅱ和Ⅲ

18. 下列寄存器中，汇编语言程序员可见的是________。

A. 存储器地址寄存器（MAR）　　B. 程序计数器（PC）

C. 存储器数据寄存器（MDR）　　D. 指令寄存器（IR）

19. 假定不采用Cache和指令预取技术，且机器处于“开中断”状态，则在下列有关指令执行的叙述中，错误的是________。

A. 每个指令周期中CPU都至少访问内存一次

B. 每个指令周期一定大于或等于一个CPU周期

C. 空操作指令的指令周期中任何寄存器的内容都不会被改变

D. 当前程序在每条指令执行结束时都可能被外部中断打断

20. 某计算机字长16位，采用16位定长指令字结构，部分数据通路结构如图6.14所示。图中所有控制信号为1时表示有效，为0时表示无效。例如控制信号MDRinE为1表示允许数据从DB打入MDR，MDRin为1表示允许数据从内总线打入MDR。假设MAR的输出一直处于使能状态。加法指令“ADD (R1,R0)”的功能为(R0)+((R1))→(R1)，即将R0中的数据与R1的内容所指主存单元的数据相加，并将结果送入R1的内容所指主存单元中保存。

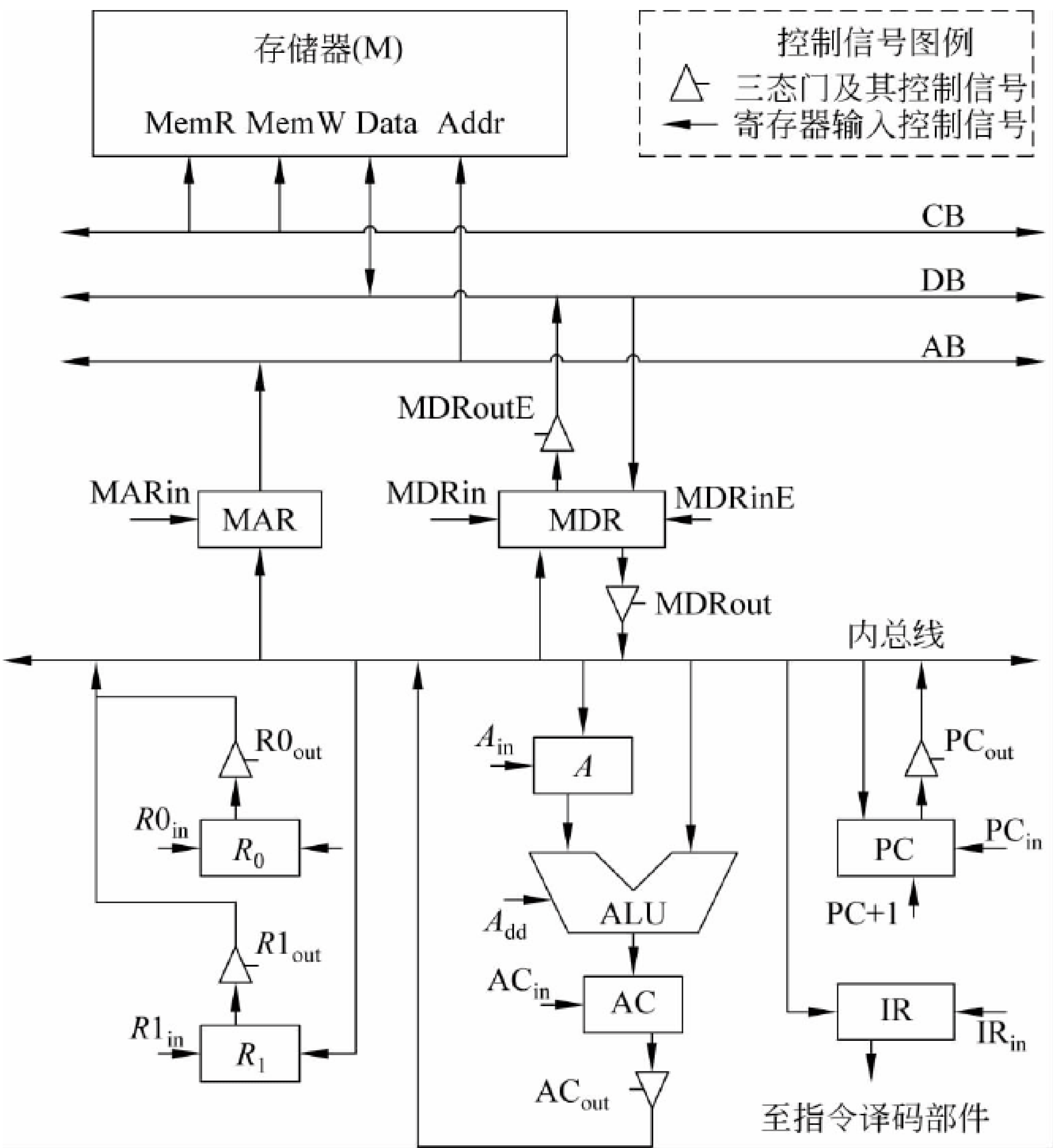


图6.14 16位字长指令结构

表6.4给出了上述指令取指和译码阶段每个节拍(时钟周期)的功能和有效控制信号，请按表中描述方式用表格列出指令执行阶段每个节拍的功能和有效控制信号。

表6.4 指令取指和译码阶段每个节拍的功能和有效控制信号

时钟	功能	有效控制信号
C1	MAR←(PC)	PCout，MARin
C2	MDR←M(MAR) PC←(PC)+1	MemR，MDRinE PC+1
C3	IR←(MDR)	MDRout，IRin

时钟	功能	有效控制信号
C4	指令译码	无

21. 某计算机字长为16位，主存地址空间大小为128KB，按字编址，采用单字长指令格式，指令各字段定义如图6.15所示。

15–12	11–9	8–6	5–3	2–0
OP	M_s	R_s	M_d	R_d
	源操作数		目的操作数	

图6.15　16位字长计算机的指令格式

转移指令采用相对寻址方式，相对偏移是用补码表示的。寻址方式定义如表6.5所示。

表6.5　相对寻址方式

M_s/M_d	寻址方式	助记符	含　　义
000B	寄存器直接	R_n	操作数＝(Rn)
001B	寄存器间接	(R_n)	操作数＝((R_n))
010B	寄存器间接、自增	(R_n)＋	操作数＝((R_n))，(R_n)＋1→R_n
011B	相对	D(R_n)	转移目标地址＝(PC)＋(R_n)

说明：(x)表示存储器地址x或寄存器x的内容。

请回答下列问题：

(1) 该指令系统最多可有多少条指令？该计算机最多有多少个通用寄存器？存储器地址寄存器(MAR)和存储器数据寄存器(MDR)至少各需多少位？

(2) 转移指令的目标地址范围是多少？

(3) 若操作码0010B表示加法操作(助记符为add)，寄存器R4和R5的编号分别为100B和101B，R4的内容为1234H，R5的内容为5678H，地址1234H中的内容为5678H，地址5678H中的内容为1234H，则汇编语句为“add (R4)，(R5)＋”(逗号前为源操作数，逗号后为目的操作数)对应的机器码是什么(用十六进制表示)？该指令执行后，哪些寄存器和存储单元的内容会改变？改变后的内容是什么？

22. 某计算机的指令流水线由4个功能段组成，指令流经各功能段的时间(忽略各功能段之间的缓存时间)分别为90 ns、80 ns、70 ns和60 ns，则该计算机的CPU时钟周期至少是_________。

A. 90 ns　　B. 80 ns　　C. 70 ns　　D. 60 ns

23. 下列不会引起指令流水阻塞的是_________。

A. 数据旁路(转发)　B. 数据相关　C. 条件转移　D. 资源冲突

第 7 章 多级结构存储器系统和主存储器

主存储器，又称内存储器，是传统计算机硬件系统的 5 大功能部件之一，用于存储处在运行中的程序和相关数据，其容量与读写速度等指标对计算机总体性能有重大影响。因此，在现代的计算机系统中，通常采用由 3 种运行原理不同、性能差异很大的存储介质分别构建高速缓冲存储器、主存储器和虚拟存储器，再将它们组成 3 级结构的统一管理调度的一体化存储器系统。由高速缓冲存储器缓解主存读写速度慢，不能满足 CPU 运行速度需要的矛盾；用虚拟存储器（快速磁盘上的一片存储区）更大的存储空间解决主存容量小，存不下更大程序与更多数据的难题。显而易见，3 级结构的存储器系统，是围绕主存储器来组织和运行的。就是说，设计与运行程序是针对主存储器进行的，充分表明主存储器在计算机系统中举足轻重的地位。

本章首先介绍多级结构存储器系统的基本组成，各级存储器所用介质的特性，多级结构存储器结构应满足的原则，以及它得以高效运行的原理；接下来讲解主存储器的设计和实用技术，包括动态存储器芯片和动态存储器芯片对一个二进制位信息的存储、读写的基本原理，组成存储器系统涉及的相关技术，提高主存器性能的思路和解决方案。

7.1 存储器系统概述

与冯·诺依曼计算机以运算器为中心不同，现代计算机系统以存储器为中心。在程序执行的过程中，CPU 调入的指令、运算器需要的数据以及需要存回的运算结果，都要对存储器进行操作；各种输入输出设备通常也直接与存储器交换数据；多处理器系统中，各处理器共享的数据通过共享存储器实现。因此，存储器是各种信息存储和交换的中心。

存储器存储一位（bit）二进制代码的存储元件称为基本存储单元。存储器中，作为一个整体存取的二进制数组成一个存储字，存储字的位数被称为存储器的字长。存放一个存储字的空间称为存储单元，按一定规则组合在一起的大量存储单元构成一个存储体。

对于存储系统，存储器与外界一次传送的二进制位数称为粒度。依据计算机计算的需要，不同的存储器或同一存储器在不同场合，与外界传送的粒度是不同的，通常有字节、字、双字、页、块、段和扇区等。

7.1.1 存储器分类

随着计算机系统结构的发展和微电子技术的进步，存储器的种类也越来越多，可以按照

多种方法对其进行分类，如按存储介质、按存取方式或按它们在计算机中的作用来划分。

1. 按存储介质划分

现代计算机中的程序和数据都是用二进制数表示的，因此，从理论上讲，只要有两个明显稳定的物理状态且状态间能比较容易地进行转换的介质，都可以用作计算机的存储器。当然，它们还必须满足一些其他要求，如读写的可靠性和速度等。

根据存储介质的不同，存储器可以划分为磁芯存储器、半导体存储器、光电存储器、磁表面存储器和光盘存储器等。当前，计算机的主存使用的大多是半导体存储器，外存一般为磁表面存储器和光盘存储器。

2. 按存取方式划分

按照不同的存取方式，存储器可以划分成随机访问存储器（Random Access Memory，RAM）、只读存储器（Read Only Memory，ROM）和顺序访问存储器（Serial Access Storage，SAS）。

从字面上讲，随机访问存储器 RAM 指的是可以通过指令随机地、个别地对各存储单元进行读写访问的存储器。一般情况下，访问所需的时间基本固定，与存储单元的地址无关。目前几乎所有的计算机中，RAM 都是主存储器的主要组成部分，一旦断电后，RAM 芯片内的已有内容会全部丢失。

只读存储器 ROM 的性质和 RAM 基本相同，只是其内容一经写入后，仅供读取使用，正常使用的情况下无法对其进行再次写入。但是断电后依然能保存已有信息，它常用来存放计算机的引导程序、各种字符及符号的字型库等内容。它和 RAM 共享计算机主存储器的地址空间，所以也是主存储器的组成部分。

ROM 并非绝对不可以写入，随着半导体技术的发展，出现了可编程只读存储器（Programmable ROM，PROM）、可擦除的可编程只读存储器（Erosible Programmable ROM，EPROM）和其他许多不同种类的 ROM。

3. 按功能效用划分

按照存储器在计算机系统中作用的不同，可将它们划分为主存储器（内存）、辅助存储器（也包括外存）和高速缓冲存储器等。主存一般由半导体存储器来承担，其特点是存取速度快，但容量较小、每位价格也比较高。外存主要是磁表面存储器和光盘存储器，如磁盘、磁带和光盘，它们容量大、每位价格低，但速度慢。高速缓冲存储器速度最高，但每位的价格也最昂贵，一般用在 CPU 和主存之间，用来缓存信息。

7.1.2 存储器系统目标

随着计算机系统的不断发展和进步，人们对计算机存储器的要求也越来越高，主要体现在以下几个方面。

1. 存储速度

微电子技术的进步带动计算机芯片的集成度每年都快速提高，使计算机行业以其他行业无法比拟的速度突飞猛进。每片芯片中集成的晶体管越多，即芯片中的逻辑门数量越多，也就意味着具有更大的主存和更强的处理能力。

虽然都以差不多的速度发展，但 CPU 和主存的发展方向却不一样。CPU 着力提高的是其处理速度，而主存的发展却是以提高容量为主，兼顾访问速度的提高。不同的发展策略

使CPU和主存的速度差距越来越大。图7.1给出了CPU和动态存储器DRAM的性能比较。

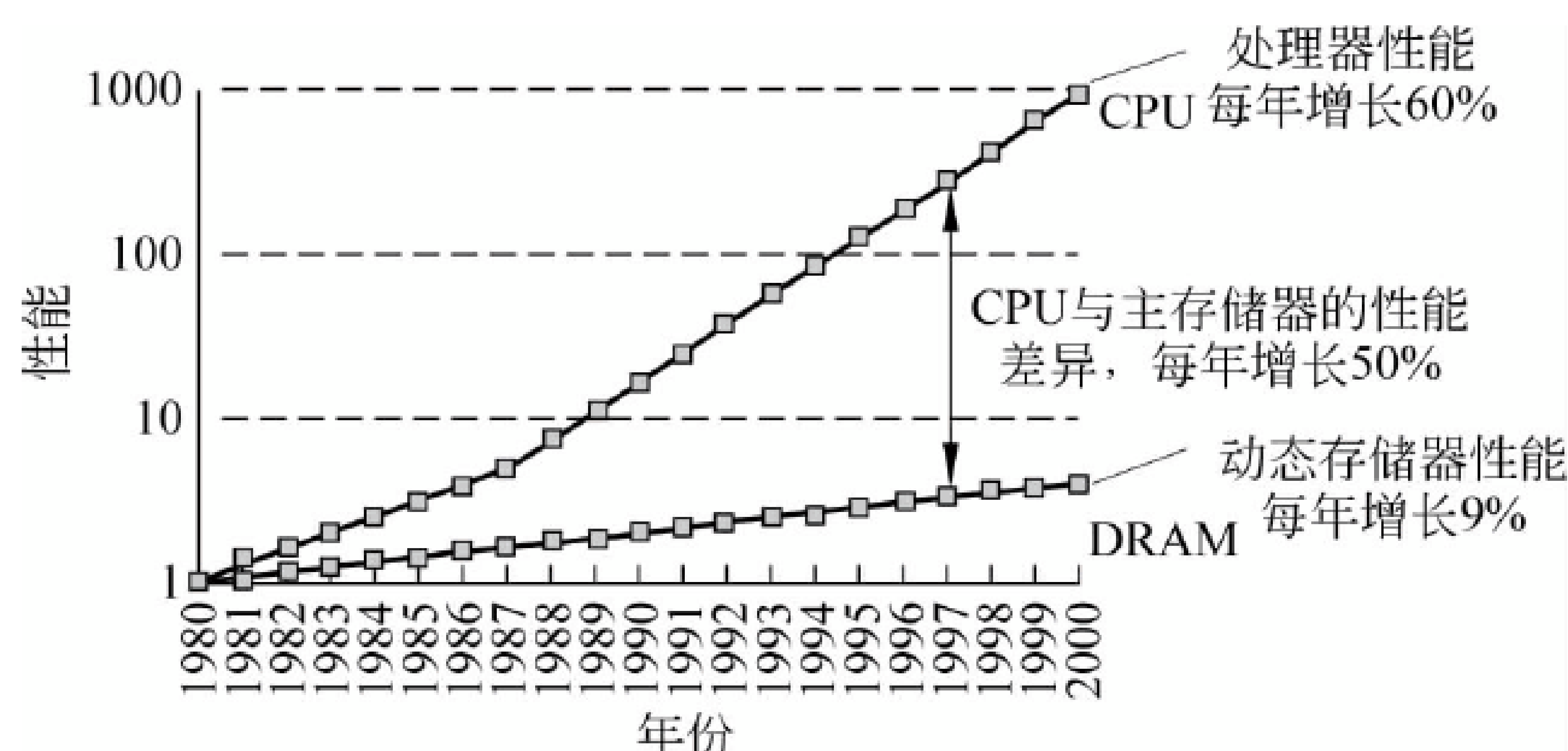


图7.1　CPU与DRAM性能比较

主存速度如果跟不上CPU速度的话，CPU也就无法发挥出其高性能。如何弥补CPU和动态存储器在性能上的差距，是存储器系统要解决的问题之一。

2. 存储容量

存储器在容量上的进步速度可以用**Moore定律**来说明，它是由Intel公司的董事长和创建者之一——Gordon Moore在1965年发现并以他的名字命名的。在为一个工业组织准备讲稿时，Moore注意到新一代主存芯片都是在它的上一代推出3年后出现的，而且都是上一代容量的4倍。Moore据此认识到每片芯片中集成的晶体管数量是以基本固定的速度增加的，而且预测今后10年也会保持这个发展速度。现在，Moore定律的通常表述是每个芯片中的晶体管数量18个月翻一番，也就是说，每年增长60%。

当然，Moore定律实际上并不是一条定律，它只是根据对固体物理学家和工艺工程师推动芯片发展工作的观察而得出的经验公式，并预测将来还会保持这个发展速度。许多工业观察家预计Moore定律在21世纪还会有效，也许到2020年。到那时，晶体管将小得无法让人相信，而且随着量子力学的发展，用单个电子的旋转来存储一位将成为可能。

然而，存储器在容量上的飞速发展依然很难满足软件对存储器的要求。微软高级经理Nathan Myhrvold提出了Nathan的软件第一定律。他认为："软件是一种可以膨胀到充满整个容器的气体。"20世纪80年代流行的字处理软件是类似于Troff之类的程序，占用几十KB主存，而现在的字处理软件要占用几十MB主存，将来毫无疑问要占用几十GB。软件的这个特点对更快的处理器、更大的主存、更强的输入/输出能力提出了持续的要求。如何满足这些要求成为存储器系统面对的又一个问题。

3. 单位价格

存储器容量的高速膨胀带来的问题是大家对其单位成本越来越高的关注。如果不考虑价格因素，也许就用不着辅助存储器了。速度和容量的增加如何控制在用户能够承受的合理价格之内，是存储器系统面临的第3个问题。

总之，存储器系统的设计需要在速度、容量和价格(/位)3个方面综合考虑，在合理的价格上提供尽可能高的访问速度和尽可能大的存储容量。

7.1.3 多级结构存储器系统

在现代计算机系统中,广泛采用由 3 种运行原理不同、性能差异很大的存储介质,来分别构建高速缓冲存储器、主存储器和虚拟存储器,再将它们组成通过计算机硬软件统一管理与调度的三级结构的存储器系统,如图 7.2 所示。

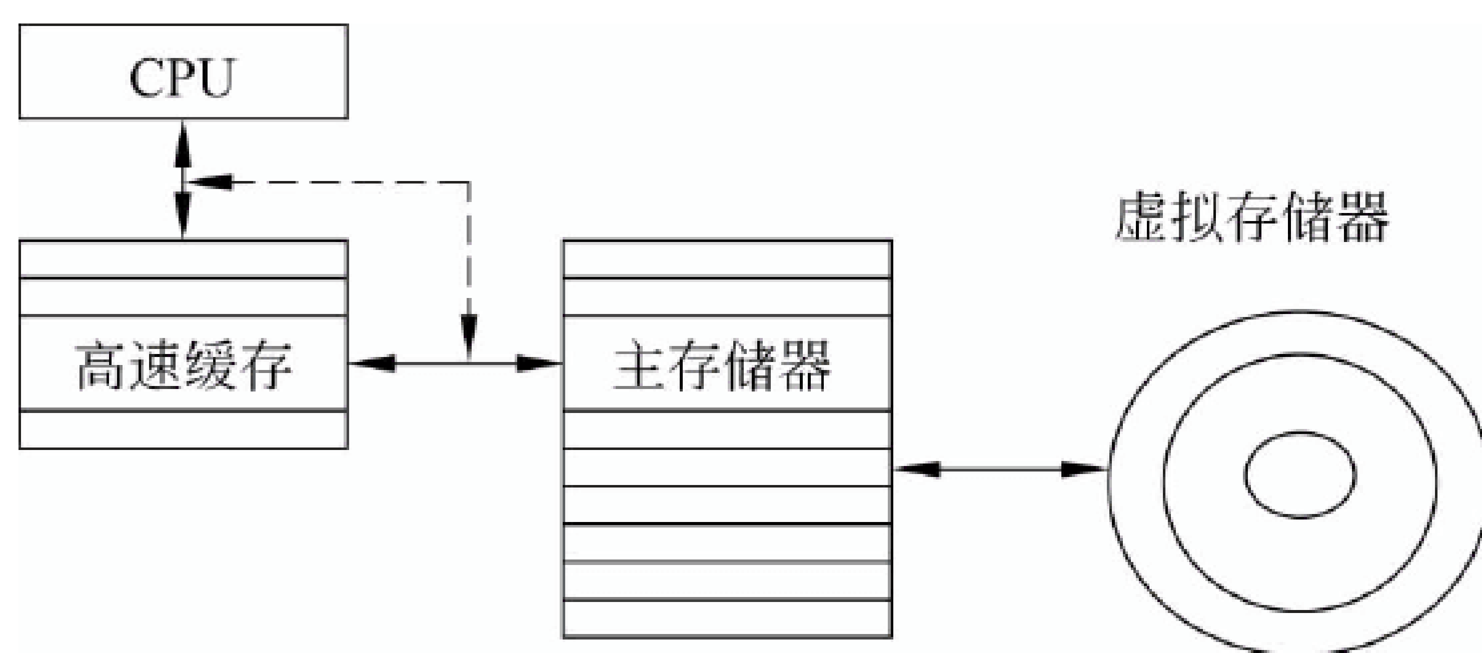


图 7.2 3 级结构的存储器系统

这种 3 级结构的存储器系统,是围绕读写速度尚可、存储容量适中的主存储器来组织和运行的,并由高速缓冲存储器缓解主存读写速度慢、不能满足 CPU 运行速度需要的矛盾;用虚拟存储器更大的存储空间解决主存容量小、存不下规模更大的程序与更多数据的难题,从而达到使整个存储器系统有更高的读写速度、尽可能大的存储容量、相对较低的制造与运行成本。高速缓冲存储器的问题是容量很小,虚拟存储器的问题是读写速太慢。追求整个存储器系统有更高的性能/价格比的核心思路,在于使用中充分发挥 3 级存储器各自的优势,尽量避开其短处。很显然,存储相同数量的信息,使用读写速度快的存储介质时,其价格通常总是要高一些。

这种 3 级结构的存储器系统的运行原理,或者说它可以有良好的性能/价格比,是建立在程序运行的局部性原理之上的。程序运行的局部性原理主要体现在如下 3 个方面。

(1) **时间方面**。在一小段时间内,最近被访问过的程序和数据很可能再次被访问。

(2) **空间方面**。这些最近被访问过的程序和数据,往往集中在一小片存储区域中。

(3) **指令执行顺序方面**。指令顺序执行比转移执行的可能性要大(大约为 5∶1)。

这样就有可能把要使用的程序和数据,按其使用的急迫和频繁程度,分时间段、分批量、合理地调入存储容量不同、读写速度不同的存储器部件中,并由计算机硬件、软件自动地统一管理与调度。就是说,把 CPU 最近一小段时间要频繁、高速使用的信息存储在高速缓冲存储器中,可以快速完成读写操作,不至于拖慢 CPU 的运行速度,问题是信息数量不能太多,但在这一小段时间内也算够用了;把在略长一段时间内要用的较多信息存放在主存储器中,只是在 CPU 从高速缓冲存储器中找不到要用的信息时,才读速度较慢的主存储器,用时会长些,但找到这一信息的概率会大得多,在把得到的信息读入 CPU 的同时,还顺便将其写入到高速缓冲存储器中,并标明这一信息来自主存储器的哪个存储单元,下次再用到这一信息时,就不必再去读速度较慢的主存储器,而是快速地从高速缓冲存储器中直接得到;把那些暂时可以先不使用的信息保存在容量非常大的虚拟存储器中,用到时再从那里以更大的批量读入主存储器,读入的速度会慢得多,但确保再大的程序和再多数据总有办法(分时、分批量地)调入主存储器且保证其正常运行。

在 3 级结构的存储器系统中,这 3 级不同的存储器中存放的信息必须满足如下两个

原则。

(1) 一致性原则。同一个信息会同时存放在几个级别的存储器中，此时，这一信息在几个级别的存储器中必须保持相同的值。

(2) 包含性原则。处在内层(更靠近 CPU)存储器中的信息一定被包含在各外层的存储器中，即内层(更靠近 CPU)存储器中的全部信息一定是各外层存储器中所存信息中一小部分的副本，这是保证程序正常运行、实现信息共享、提高系统资源利用率所必须的，反之则不成立。例如，高速缓冲存储器中的信息，肯定也存放在主存储器中，还存放在虚拟存储器中，但主存储器中的非常多的信息不会同时在高速缓冲存储器中，虚拟存储器中的更多的信息也不会同时出现在主存储器中。

3 级不同的存储器是用读写速度不同、存储容量不同、运行原理不同、管理使用办法也不尽相同的不同存储介质实现的。高速缓冲存储器使用静态存储器芯片实现，主存储器通常使用动态存储器芯片实现，而虚拟存储器则使用快速磁盘设备上的一片存储区。前两者是半导体电路器件，以数字逻辑电路方式进行读写，后者则是在磁性介质层中通过电磁转换过程完成信息读写。

7.2 主存储器

7.2.1 主存储器概述

主存储器是计算机硬件系统中的 5 大功能部件之一，用于存放正在运行中的程序和相关数据。它的读写速度和存储容量对计算机系统的运行性能有至关重要的影响，经常成为影响系统运行性能的瓶颈。读写速度通常用读、写一个存储单元必需的时间度量，例如 60ns，连续两次读写必需的时间间隔被称为存储周期，考虑到线路恢复的延迟问题，它应略大于一次主存读、写所用的时间。存储容量通常用构成存储器的字节数或字数表述，一个存储字通常由 2、4、8 个字节组成。多数计算机都能在逻辑上同时支持按字或字节读写主存储器。

主存储器通过地址总线、数据总线、控制总线与计算机的 CPU 和外围设备连接在一起，如图 7.3 所示。

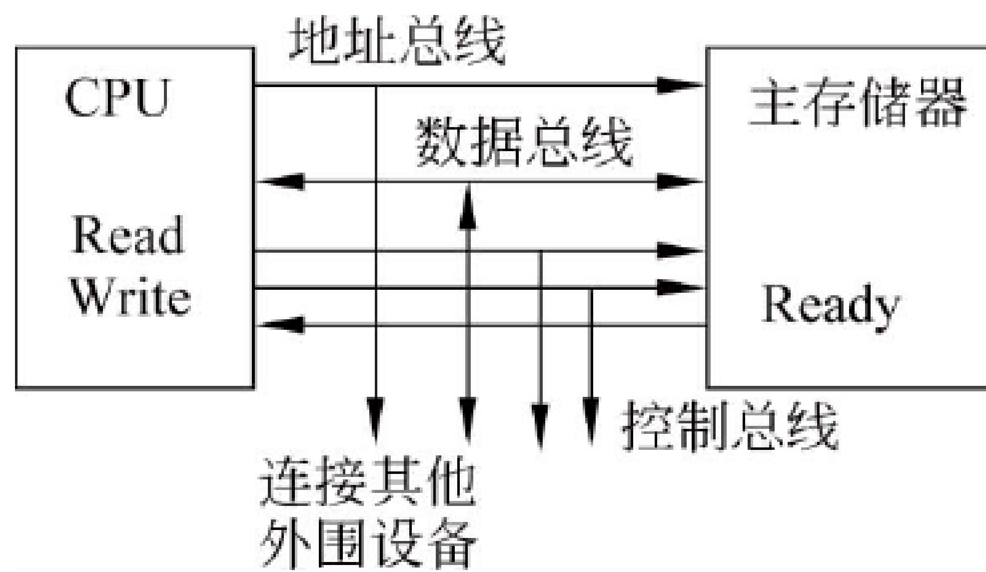


图 7.3 主存储器与其他部件的连接关系

(1) 地址总线用于选择主存储器的一个存储单元(字或字节)，其位数决定了可以访问的存储单元的最大数目，称为最大可寻址空间。例如，当按字节寻址时，20 位的地址可以访问 1MB 的存储空间，32 位的地址可以访问 4GB 的存储空间。

(2) 数据总线用于在计算机各功能部件之间传送数据，数据总线的位数(总线的宽度)与总线时钟频率的乘积，正比于该总线所支持的最高数据吞吐(输入输出)能力。

(3) 控制总线用于指明总线的工作周期类型和本次入/出完成的时刻，总线的工作周期可以包括主存储器读周期、主存储器写周期、I/O 设备读周期、I/O 设备写周期，即区分要用哪个部件(主存或 I/O 设备)和操作的性质(读或写)；还有直接存储器访问(DMA)总线周期

等。若在计算机系统中使用了不同读写速度的主存储器，在 CPU 发出读写主存储器的命令后，它不能知晓读写操作完成的时刻，这是由被读写的存储器(或外围设备)本身的运行速度决定的，此时可以让主存储器本身提供读写完成的回答信号(Ready)，CPU 通过检测该信号来得知本次读写完成的时刻；若为读操作，有了该回答信号后，CPU 就可以接收已读出的数据，如图 7.3 所示，这被称为 CPU 和主存储器按异步方式运行。

RAM 存储器芯片中，通常由存储阵列、译码器、读写控制电路和数据缓冲电路等部分组成，如图 7.4(a)所示。其中的存储阵列由大量相同的基本存储单元阵列构成；译码器电路将来自 CPU 输入的地址信号翻译成某单元(存储字或字节)的选通信号，使该单元能够被读写；控制电路对存储芯片进行芯片控制、读写控制和输出控制等操作，它们分别通过$\overline{CE}$(Chip Enable)、$\overline{WE}$(Write Enable)和$\overline{OE}$(Output Enable)引脚实现；缓冲电路用于寄存来自 CPU 的写入数据或从存储体内读出的数据，具有三态控制。注意，当芯片控制$\overline{CE}$无效，其他所有的信号都不起作用，即芯片不能工作；通常$\overline{OE}$信号与$\overline{WE}$信号是互斥的，即进行读操作时，$\overline{OE}$信号为低，而$\overline{WE}$信号为高。

存储器芯片的译码方式分为单译码和双译码，通常大容量存储器采用双译码方式。图 7.4(b)为双译码方式结构的存储器的示意，双译码方式可以节省大量的选通线，例如 10 条地址线，单译码方式需要 2^{10} 共 1024 条选通线，而双译码方式仅需要 2^5+2^5 共 64 条选通线。

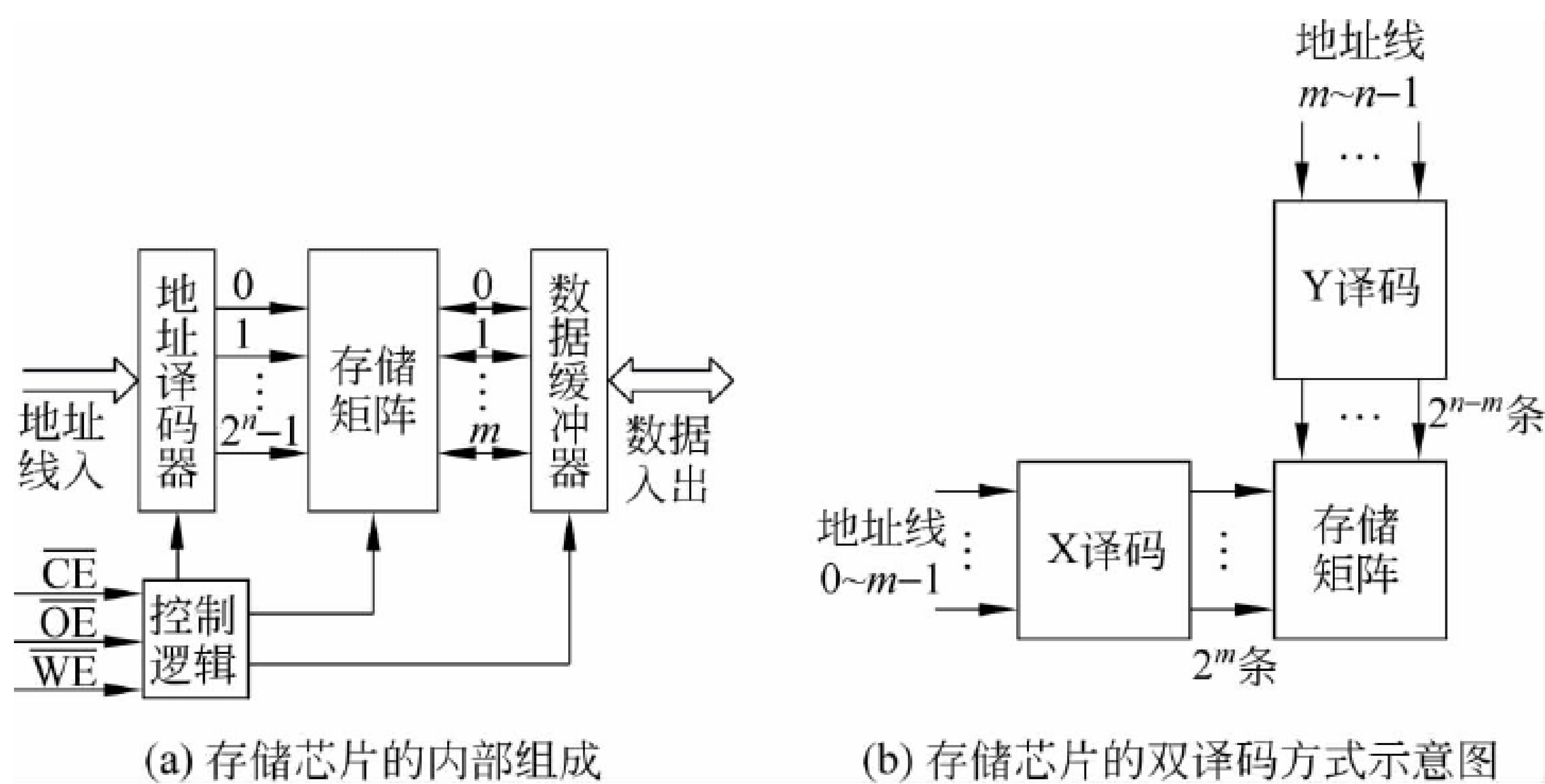


图 7.4　存储器芯片的内部组成与双译码示意图

从所用的半导体生产工艺区分，RAM 存储器芯片又可以分为静态存储器(SRAM)和动态存储器(DRAM)两种类型。由于动态存储器集成度高、生产成本低，被广泛地用于实现要求更大容量的主存储器。静态存储器读写速度快、生产成本高，通常多用其实现容量可以较小的高速缓冲存储器。静态存储器和动态存储器的存储原理在下面详细介绍，它们的不同之处如表 7.1 所示。

表 7.1　静态存储器与动态存储器的区别

	SRAM	DRAM
存储信息	触发器	电容
破坏性读出	非	是
需要刷新	非	需要

续表

	SRAM	DRAM
行列地址	同时送	分两次送
运行速度	快	慢
集成度	高	低
发热量	大	小
存储成本	高	低

7.2.2 动态存储器的存储原理

动态RAM的工作原理如图7.5(a)所示,是由MOS管的栅源电容C_s来存储1位二进制信息的,并用一个MOS管T来控制数据的读写,C_s中存有电荷表示1,无电荷表示0。它的特点是用较少的晶体管构成一个存储单元,由此提高芯片单位面积上的容量,同时也降低了每位价格和功耗。

当字选择线为低电平时,MOS管T截止,电容C_s上有无电荷的情况不会反映到T的另一端;当字选择线为高电平时,MOS管T导通,电源V_{DD}经电容与位线连通,依据位线上的电位是高还是低,以及电容C_s上有无电荷的不同组合情况,MOS管T中会呈现电流有无流过两种不同情况。

对于单管MOS动态存储单元,由于C_s的容量很小,电路中不可避免地存在漏电流,通常存储的信息只能保持若干个毫秒,再由于动态RAM是破坏性读出,所以必须对它进行**刷新**。注意,刷新和读出都由刷新和读出放大电路完成,如图7.5(b)所示。读出时,当列选择线有效,数据才通过T_2送至芯片的数据引脚I/O;刷新时,将同一行各存储单元的信息读入刷新和读出放大电路,再写回原存储单元,与列选择线无关。通常有两种刷新方式。

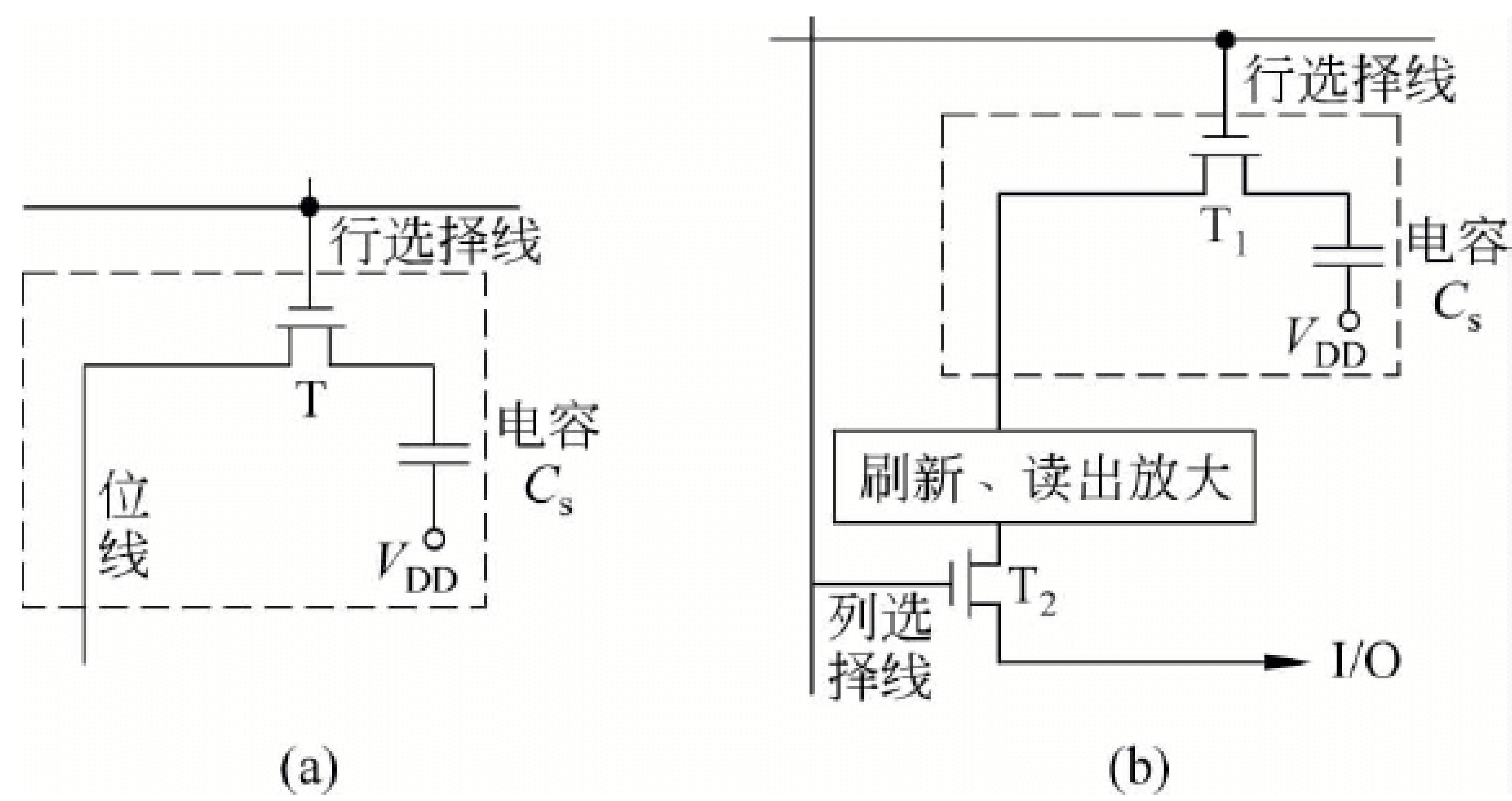


图7.5 MOS单管动态存储器的组成

1. 集中式刷新

集中式刷新指在一个刷新周期内,利用一段固定的时间,依次对存储器的所有行逐一再生,在此期间停止对存储器的读和写。

例如,一个存储器有1024行,系统工作周期为200ns。RAM刷新周期为2ms。这样,在每个刷新周期内共有10000个工作周期,其中用于再生的为1024个工作周期,用于读和

写的为8976个工作周期。

集中刷新的缺点是在刷新期间不能访问存储器。

2. 分散式刷新

分散式刷新有两种方法。

第一种，把对每一行的再生分散到各个工作周期中去。这样，一个存储器的系统工作周期分为两部分：前半部分用于正常读、写或保持；后半部分用于再生某一行。系统工作周期增加到400ns，每1024个系统工作周期可把整个存储器刷新一遍。可以看出，整个存储器的刷新周期缩短了，它不是2ms，而是409.6μs。但由于它的系统工作周期为读、写所需时间周期的2倍，存储器不能高速工作，在实际应用时要加以改进。

第二种，为了提高存储器工作效率，经常采取在2ms时间内分散地将1024行刷新一遍的方法，具体做法是将刷新周期除以行数，得到两次刷新操作之间的时间间隔 t，利用逻辑电路每一时间间隔 t 产生一次刷新请求。

动态MOS存储器的刷新需要有硬件电路的支持，包括刷新计数器、刷新/访存裁决、刷新控制逻辑等。

由于刷新与列地址无关，同时为了减少地址线引脚和节省动态RAM的体积，所以动态RAM采用**地址复用**的方法，将地址分为**行地址**和**列地址**，并且分别由**行地址选通 $\overline{\text{RAS}}$** 和**列地址选通 $\overline{\text{CAS}}$** 控制。例如，Intel的2164A为64K×1的DRAM，图7.6为芯片的引脚图，芯片用8根复用地址线，分两批传送16位地址。

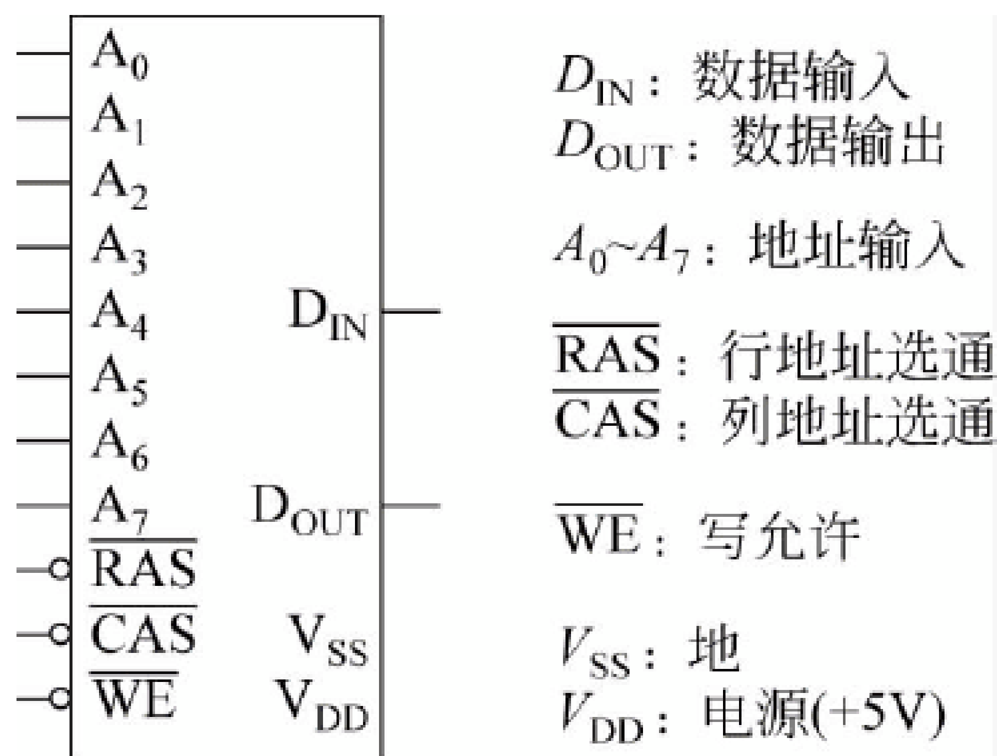


图7.6　2164A动态RAM的引脚

动态存储器读写过程中，行、列地址的建立时间，读写信号的建立时间，得到读出数据、提供写入数据的时间之间，要满足规定的条件(技术指标)。限于篇幅，我们无法在这里详细讨论这些线路与信号时序问题，只是给出一些结论。

CPU向动态存储器送地址是一次完成的，由存储器芯片内部的行地址和列地址锁存器线路分先后接收行、列地址；执行读操作时，为了保证正常读出数据，应在列地址锁存前建立读写命令信号 $\overline{\text{WE}}$(为高电位)，并在列地址锁存信号撤销后再结束 $\overline{\text{WE}}$ 信号，读出数据送到 D_{OUT} 引脚。

执行写操作时，读写命令信号 $\overline{\text{WE}}$(为低电位)，应在列地址锁存前建立，写入数据 D_{IN} 也应在列地址锁存前建立，$\overline{\text{WE}}$ 和写入数据应保持一定的时间。

动态存储器是破坏性读出，读操作之后必须接着执行写操作，此时 $\overline{\text{WE}}$ 为先高(执行读)后低(执行写)，这一跳变时间要保证读、写操作的正常执行，如图7.7所示。

7.2.3　静态存储器的存储原理

典型的1位MOS静态RAM单元如图7.8所示，它由6个MOS管组成。

MOS管 T_1 和 T_2 的输入、输出交叉耦合组成双稳态触发器，用于记忆1位二进制信息。电路中 T_3、T_4 起着负载电阻的作用，例如，T_1 管导通(T_2 管一定处于截止)，输出端为

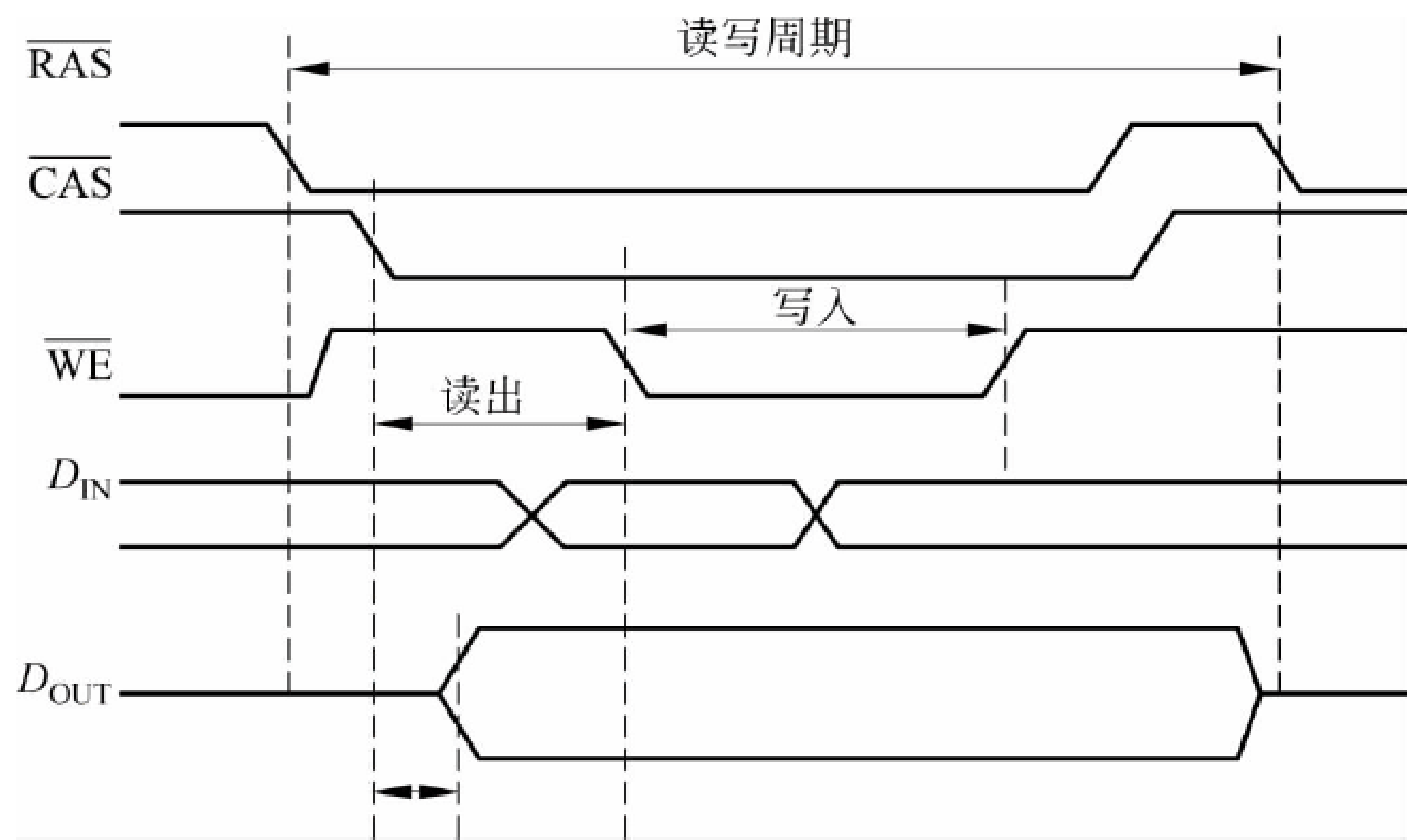


图 7.7　动态存储器的读写工作方式时序波形

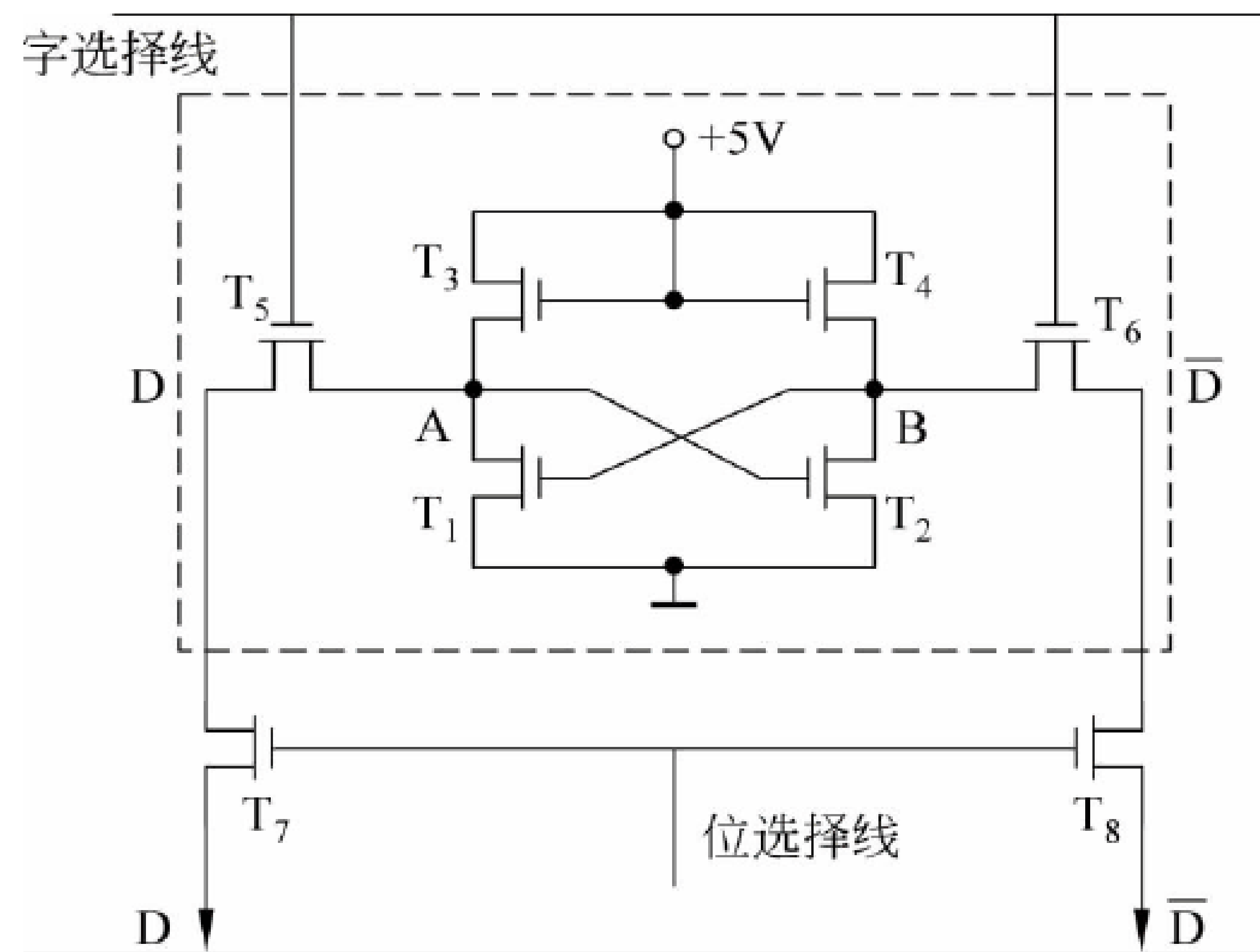


图 7.8　1 位 MOS 存储单元的组成

低,则存储 0 信号;反之,T_2 导通,存储的是 1 信号。T_5 和 T_6 能使触发器与外部电路连通或隔离,连通时兼传送读写的数据信号,由**字选**(也称**行选**)信号控制;**位选**(也称**列选**)信号控制 T_7 和 T_8,使 T_5 和 T_6 传送的信号传至芯片的数据引脚 I/O。T_7、T_8 是一列共用的,图中 T_7、T_8 下面的 D 和 $\overline{D}$ 也称位线 1 和**位线** 2。

大量的 1 位存储单元可组成容量更大的存储器芯片,例如 2048×1 的芯片,需将它们组织成 64×32 的矩阵形式,每个存储单元被连接到不同的字选线、位选线的交叉点处,并加进读写控制电路,用地址译码器提供字、位选择信号,如图 7.9 所示。由该图可知,当输入某地址信号,使对应的字、位选择线有效,就能对相应的 1 位存储单元进行读或写。显然,要组成 2048×8 的芯片,只要将 8 个相同的芯片作适当地叠加,并将数据信号增加到 8 位,不需要增加地址译码信号。

7.2.4　存储器容量扩展

一般存储器芯片的容量是有限的,它在字数或字长方面与实际存储器的要求都有很大差距。在构成主存储器的存储体时,需要在字向和位向两方面进行扩充才能满足实际主存储器的容量需要。

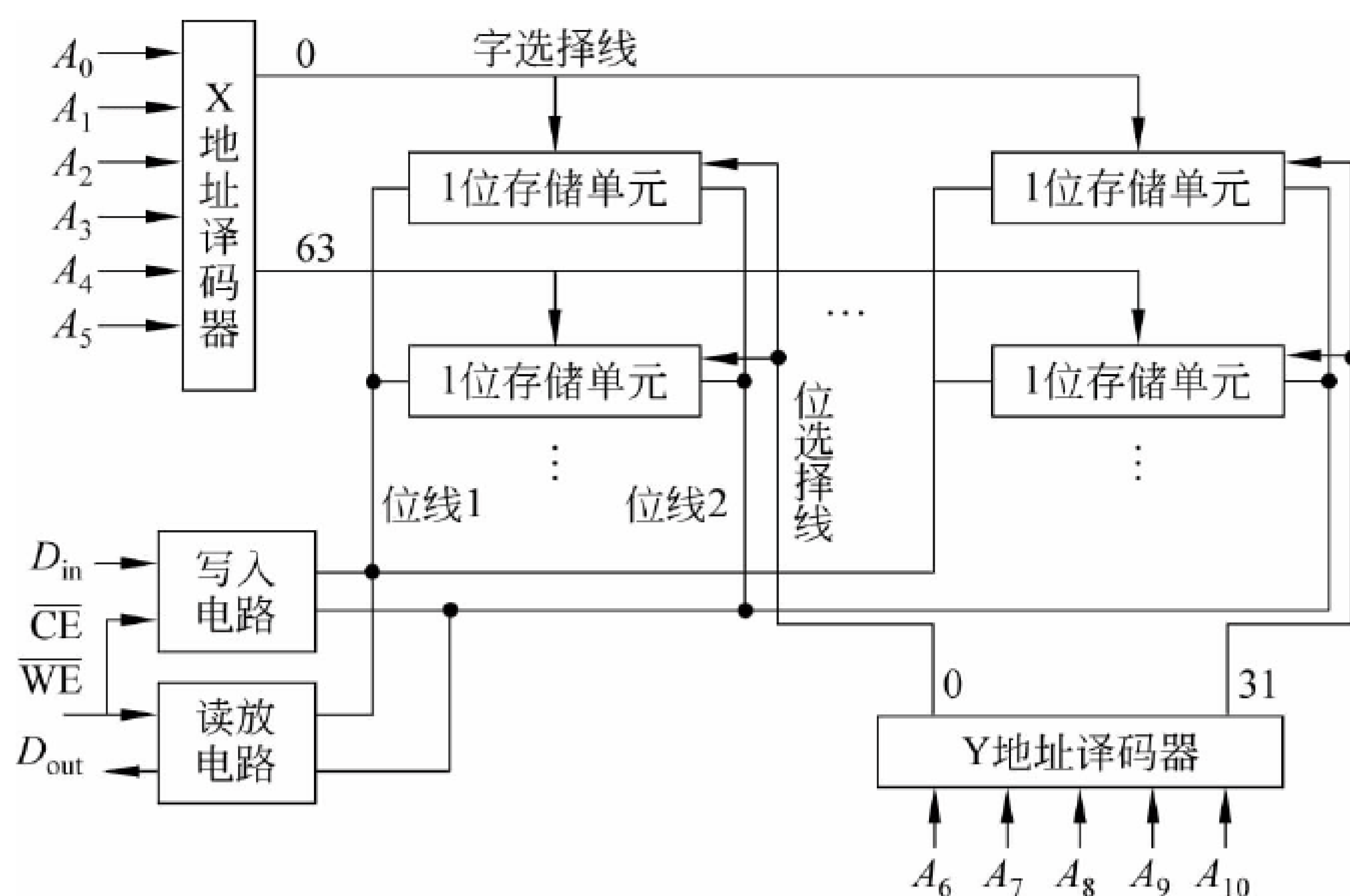


图 7.9　2KB 静态存储器芯片的逻辑组成

1. 位扩展

位扩展指的是加大字长。位扩展的连接方式是将多片存储器的地址、片选、读/写端连接在一起，数据端单独引出。如图 7.10 所示的位扩展方式是用 8 个 16K×1 位芯片组成 16K×8 位的存储器。图中每个芯片字长是 1 位，存储器字长 8 位，因此它由 8 片芯片并联而成。每片有 14 条地址线引出端，每条地址线接有 8 个芯片；每片有 1 条数据线引出端，每条数据线连接一个芯片。在位扩展时，没有选片的要求，如果芯片有片选信号($\overline{CS}$)，将它们直接接地即可。

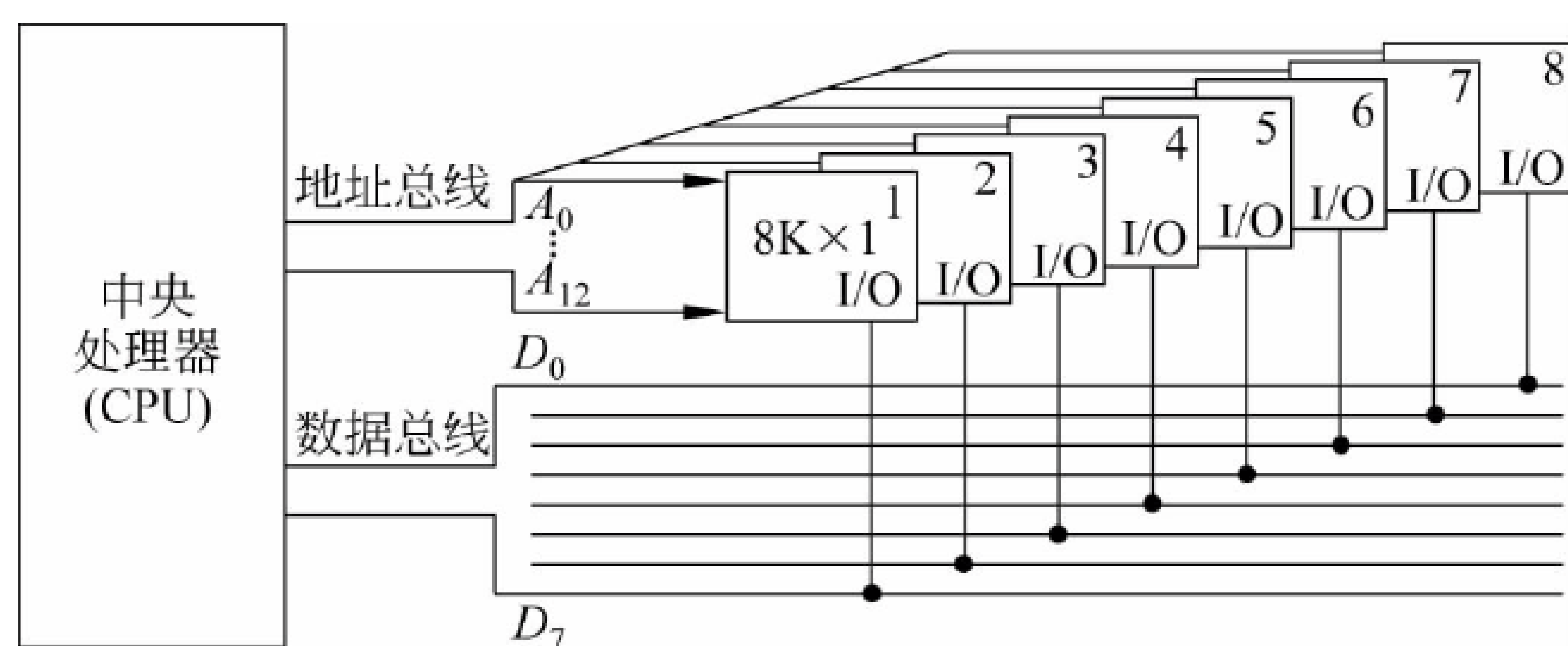


图 7.10　位扩展连接方式

2. 字扩展

字扩展指的是增加存储器中字的数量。静态存储器进行字扩展时，将各芯片的地址线、数据线、读/写控制线连接在一起，而由片选信号来区分各芯片的地址范围。如图 7.11 所示的字扩展存储器是用 4 个 16K×8 位芯片组成的 64K×8 位存储器。数据线 $D_0 \sim D_7$ 与各片的数据端相连，地址总线低位地址 $A_0 \sim A_{13}$ 与各芯片的 14 位地址端相连，而两位高位地址 A_{14}、A_{15} 经过译码器和 4 个芯片的片选端相连。动态存储器一般不设置$\overline{CS}$端，但可用$\overline{RAS}$端来扩展字数。

3. 字位扩展

实际存储器往往需要字向和位向同时扩充。一个存储器的容量为 $M \times N$ 位，若使用

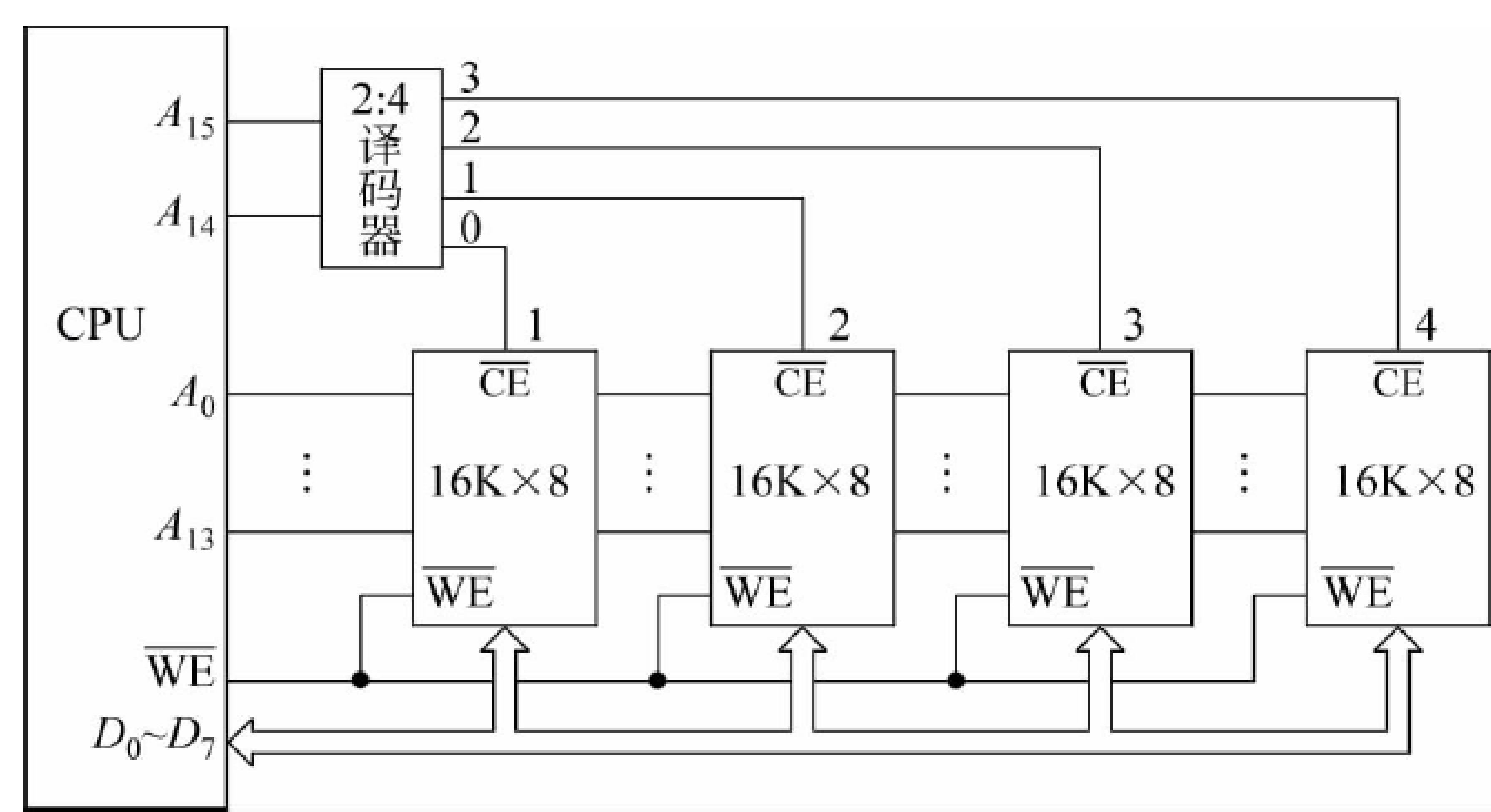


图 7.11　字扩展连接方式

$L \times K$ 位存储器芯片，那么，这个存储器共需要$\frac{M}{L} \times \frac{N}{K}$个存储器芯片。

一个小容量存储器与 CPU 的连接方式如图 7.12 所示。存储器由 Intel 2114 芯片经字位扩展而成，容量为 4K×8 位。由于 Intel 2114 芯片只有 1K×4 位，所以整个存储器共需$\frac{4}{1} \times \frac{8}{4} = 8$ 个 2114 芯片。Intel 2114 芯片本身共有 10 个地址端($A_0 \sim A_9$)、4 位数据端($D_0 \sim D_4$)、一个片选端($\overline{CS}$)和一个读写控制信号端($\overline{WE}$)。CPU 提供 12 位地址，其中低 10 位($A_0 \sim A_9$)并行连接各芯片的地址端，还有两位地址(A_{10}、A_{11})连向译码器，产生 4 个片选信号，分别控制 4 组芯片。此处译码器要受 CPU 的访存信号$\overline{MREQ}$控制，只在需要访问主存时才产生译码输出。CPU 提供 8 位数据总线($D_0 \sim D_7$)，每根数据线连接 4 个芯片。

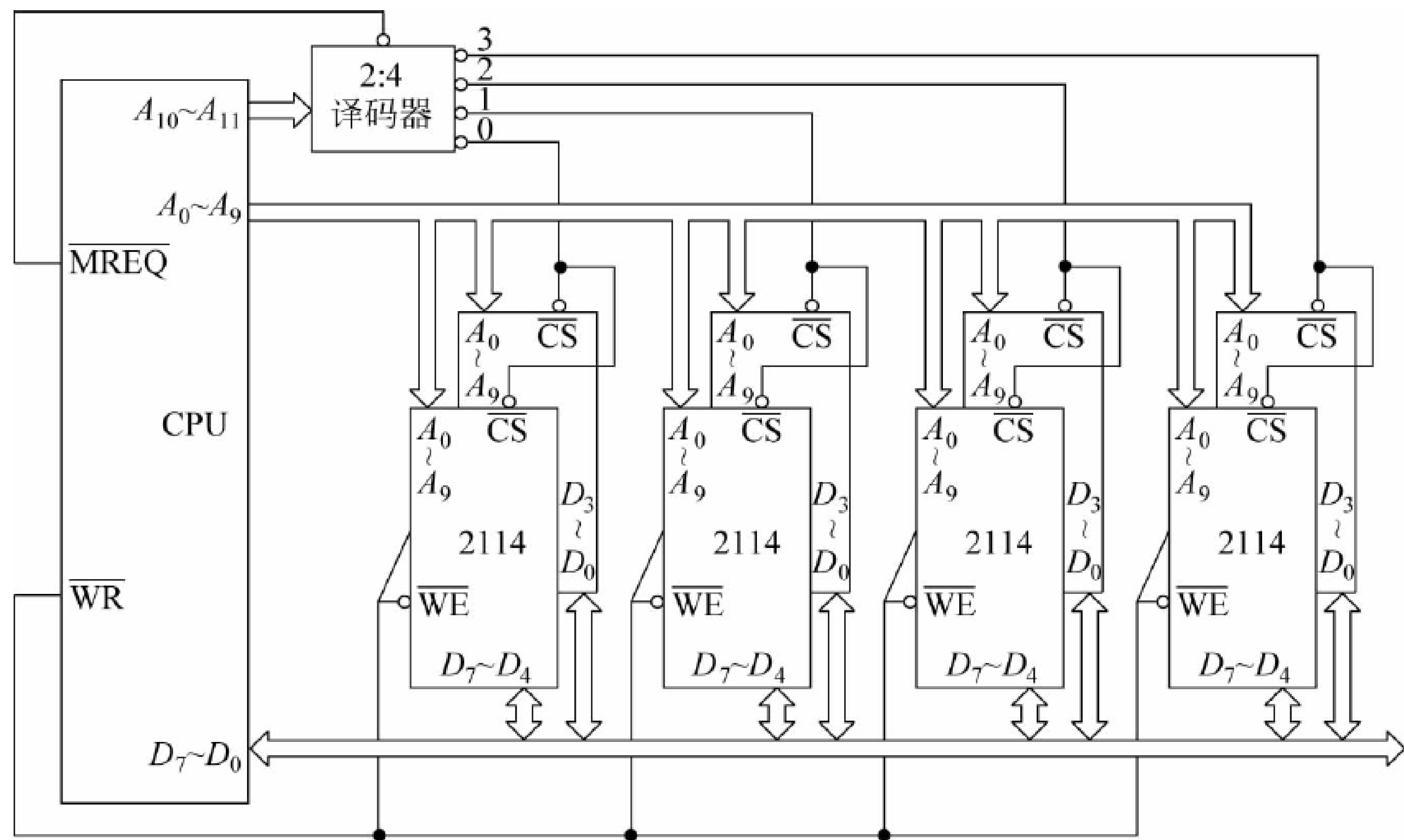


图 7.12　字位扩展连接方式

7.3 教学计算机的主存储器实例

教学计算机的基本主存储器用由 8K 字的 ROM 区和 2K 字的 RAM 区组成。主存字长 16 位，容量总共 10K 字，按字寻址方式读写。教学计算机选用单总线结构，即 CPU（控制器和运算器）、主存、串行接口都直接连接到唯一的一组总线，仿真终端通过串行接口接入主机系统。

ROM 区选用 8K×8 位的 E^2PROM 芯片 58C65 组成，RAM 区选用 2K×8 位的 SRAM 芯片 6116 组成。

用两片 58C65 芯片进行位扩展，组成 8K 字的 ROM 存储区，地址分配在 0000H～1FFF H（十进制的 0～8191）范围内；用两片 6116 芯片进行位扩展，组成 2K 字的 RAM 存储区，地址分配在 2000H～27FFH（十进制的 8192～10239）范围内。对 ROM 区和 RAM 区完成字扩展，构成 10K 的主存储区。主存储器的结构如图 7.13 所示。

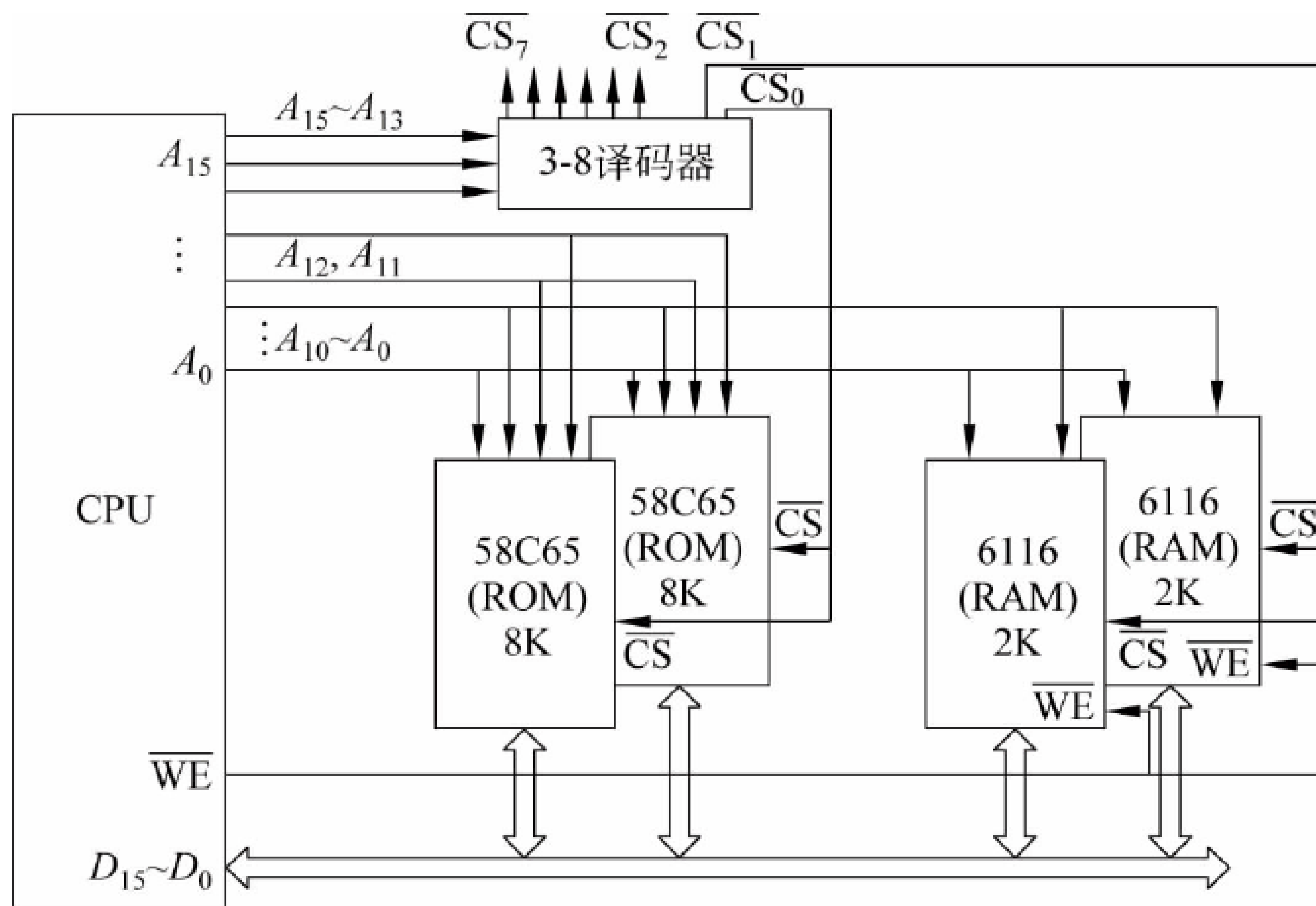


图 7.13　教学计算机主存储器结构示意图

1. 地址总线

地址总线（记为 A_{15}～A_0）的信息统一由 CPU 提供，可以来自程序计数器 PC 或内存地址寄存器 AR，而地址寄存器只接收由 ALU 输出的信息。这是由教学计算机的指令系统和硬件结构特点共同决定的。地址总线的最高 3 位送往一个译码器线路，用以产生主存芯片的片选信号，低 13 位送到每个 58C65 芯片，用于选择芯片内的一个字单元，因此每个片选信号对应着一个 8K 容量的存储空间。6116 芯片只有 2K 容量，只有 11 个地址引脚，不会用到地址总线的 A_{12} 和 A_{11} 这两位。

教学计算机选用 8 位的 I/O 端口地址，由指令寄存器 IR 的低 8 位提供，2 路串口的端口地址分别为十六进制的 80、81 和 90、91。I/O 端口地址的高 4 位送到一个译码器线路，用于产生 I/O 接口芯片的片选信号，低 4 位将按接口芯片的要求送到芯片的相应引脚，以选择接口内不同的寄存器。

2. 控制总线

控制总线信号经一片双2-4译码器74LS139和两片3-8译码器芯片74LS138给出。控制总线的基本功能，是用来指明总线周期的类型和本次入/出操作完成的时刻。教学计算机中的基本总线周期类型包括主存写、主存读、外设(接口)写、外设(接口)读4类，分别用$\overline{\text{MMW}}$、$\overline{\text{MMR}}$、$\overline{\text{IOW}}$、$\overline{\text{IOR}}$这4个信号标记，并用$\overline{\text{MMREQ}}$和$\overline{\text{IOREQ}}$来区分是主存工作还是外设工作。这些信号的产生是通过3位控制码来确定的。更深一步说，这3位控制码又是依据主存读写、I/O读写操作的需求来赋值的，即$\overline{\text{MIO}}$、REQ、$\overline{\text{WE}}$取值为000、001、010、011、1XX时，分别表示主存写、主存读、I/O写、I/O读、无读写(NC)功能。

为简化设计与实现，主存储器和CPU以同步方式交换信息，可在一个时钟周期完成；执行输入输出操作选用程序直接控制方式，需要CPU通过查询串行接口的运行状态来实现与输入输出设备(仿真终端)的同步。

3. 数据总线

数据总线是在计算机各功能部件之间完成数据传送的线路。它的工作速度(总线时钟频率)和位数(宽度)的乘积决定了总线上最大的数据传送率。

在教学计算机中数据总线16位，由于系统采用的是单总线结构，因此运算器、控制器、主存储器和串行接口的数据入出引脚都直接连接到数据总线DB，包括用于向教学机提供调试用数据的开关(经过带有三态控制的74LS244芯片)的输出也需要连接到DB。

设计和使用总线的核心技术，是要保证在任何时刻只能把一组数据发送到总线上，但允许一到多个部件同时接收总线上的信息。为此，对多个申请发送数据到总线上的部件，必须进行选择控制，确保它们以分时方式共享总线。所用的电路通常为带有高阻态输出的三态门电路，如图7.14所示。输入端标记为In，输出端标记为Out，$\overline{\text{C}}$为控制端。当$\overline{\text{C}}$为低电平时，该线路的功能是Out＝In；当$\overline{\text{C}}$高电平时，输出Out则为高阻态。

我们很容易用这样的线路构建数据总线。如图7.15所示。

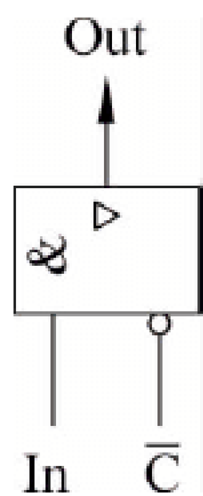


图7.14　三态门电路图

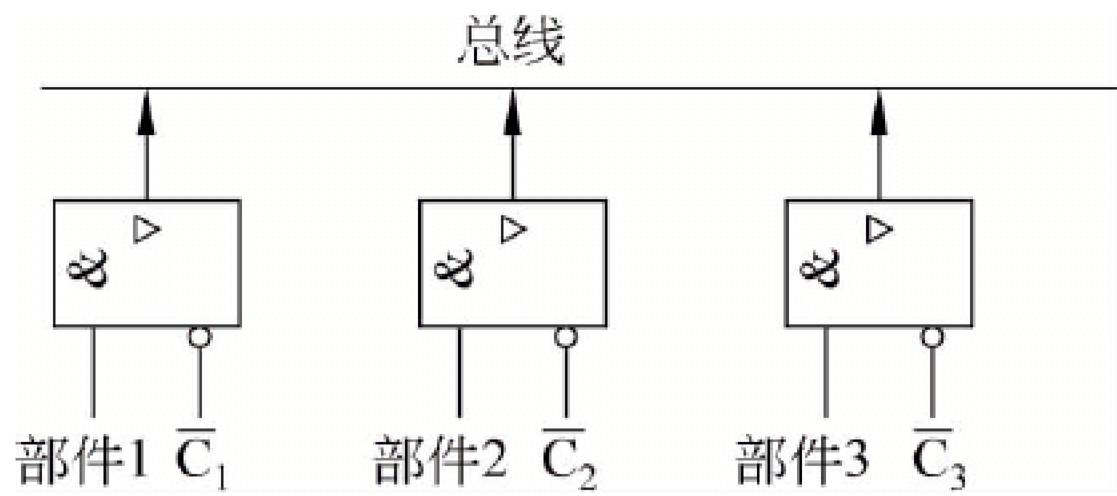


图7.15　选用三态门构建总线

把不同部件的输出经三态门连接到总线上，通过一组控制信号$\overline{\text{C}}_i$(i＝1,2,3,…)选择送入总线的数据。若做到在任何时刻，在多个控制信号中只有一个为低电平，而其他信号均为高电平，就能确保了多个部件能在不同的时刻分别向总线发送数据。若有多于1个的控制信号同时为低电平，则是严重错误，会造成多个缓冲门的输出短路，使总线上的信息混乱，甚至烧坏器件。为此，用译码器线路给出这类控制信号是一种比较理想的方案。

从总线上向外传送数据，重点是解决线路的负载与驱动能力问题。由于总线驱动的部件可能较多，又往往有较长的传送路径，要求总线所用器件有比较强的驱动能力。

4. 系统时钟及时序

计算机各功能部件都是在时钟信号的“驱动”下一步一步地协调运行并执行各自的功

能。从系统运行速度上考虑，总是希望在系统能正常运行的前提下，使系统时钟频率尽量高一些。

教学计算机无意追求运行速度。从支持串行接口芯片 Intel 8251 所要求的时钟频率考虑，教学计算机选用了 1.8432MHz 的晶振，6 分频后用307.2kHz 的时钟作为系统主时钟(也可以选择另外几个频率)，其波形如图 7.16 所示，既用于 CPU，也用于 I/O 总线，并保持 CPU 与主存、串口读写操作的同步运行。

CPU 内部的某些寄存器，通常用该时钟信号结束时的上升沿完成接收操作(边沿控制方式)，这意味着每个时钟脉冲时间对应一条微指令的时间，即一个微周期，或称一个 CPU 周期。只有运算器内的通用寄存器是用时钟脉冲的低电平接收输入(电平控制方式)。

在执行主存或 I/O 读写操作时，每个总线周期通常由两个时钟周期组成。第一个时钟周期被称为地址时间，用于传送主存地址或 I/O 端口地址；第二个时钟周期被称为数据时间，用于读写数据。由于所选的时钟频率相当低，一次数据时间足以完成主存读写操作，这就是常说的在总线的零等待状态完成数据读写。在另外一些计算机中，若一次数据时间不能完成主存或 I/O 设备的读写操作，就可以增加 1 到多个数据时间(多个时钟脉冲时间)，这增加了的数据时间被称为总线的等待状态，它降低了系统的输入输出能力。在教学计算机系统中，仿真终端的输入输出操作以程序直接控制方式运行，会用掉很多 CPU 时间。

综上所述，存储器读写涉及地址信息、读出与写入的数据内容、片选信号和读写操作命令信号，它们之间在时序配合上要满足某些条件。例如，有了稳定的地址与片选信号才可以读，有了稳定的地址和写入的数据后，有了片选信号才能再给出写命令，以便保证无误的写操作。此外，这些信号应有一定的持续时间，以保证读写操作得以正常完成。

7.4　提高主存储器性能的途径

1. 动态存储器的快速读写技术

读写动态存储器时，通常需要为存储器芯片先后分别锁存行地址和列地址，比较费时间，如果连续读写属于同一行的多个列中的数据，其行地址只需在第一次读写时送入(锁存)，之后保持不变，则每次读写属于该行的多个列中的数据时，每次仅锁存列地址即可，从而省掉了锁存行地址时间，也就加快了主存储器的读写速度。这一技术被称为动态存储器的快速读写技术，或称为快速页式工作技术，这是动态存储器所特有的用法。

2. 主存储器的并行读写技术

主存储器的并行读写是指在主存储器的一个工作周期可 多个主存字所采用的技术，在静态或动态存储器中均可使用。

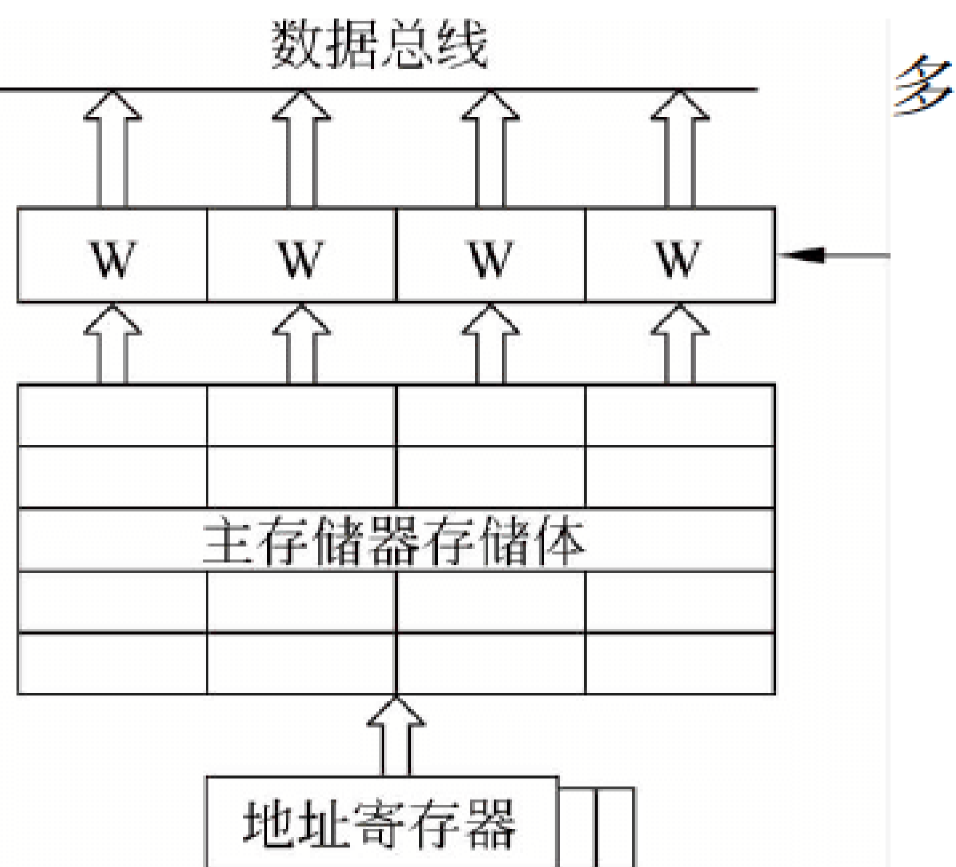


图 7.17 主存的一体多字结构

第一种方案是一体多字方案，通过加宽每个主存单元的宽度，即增加每个主存单元所包括的数据位数，使每个主存单元同时存储几个主存字，则每一次读操作就同时读出了几个主存字，使读出一个主存字的平均读出时间变为原来(与每个单元存一个字相比)的几分之一。其缺点是，每次读出的几个主存字必须首先保存在一个位数足够长的寄存器中，等待通过数据总线分几次把它们传送走。图

7.17给出了这一方案的示意表示。

第二种方案是更加常用的办法，称为多体交叉编址技术，把主存储器分成几个能独立读写、字长为一个主存字的主体，这样就可以按读写需要情况，分别对几个存储体执行读写，通过合理的组织方式，使几个存储体协同运行，从而提供出比单个存储体更高的(几倍)读写速度。合理地对这多个存储体进行组织，涉及如下两个问题。

首先是怎么对这些存储体执行读写，可以用两种方式进行处理。一是在同一个读写周期同时启动所有存储体的读或写操作，这与前面讲的一体多字方案很类似；二是使这些存储体顺序地轮流启动各自的读写周期，理论上能达到的最高读写速度，是在一个读写周期内，能启动每一个存储体的读写操作，即启动相邻两个存储体的最小时间间隔，要小于或等于一个读写周期除以存储体的个数。这种方案的优点，是依次读出来的每一个存储字，可以直接通过数据总线依次传送走，而不必设置专门的数据缓冲寄存器。

其次是如何分配这些存储体各自工作的地址范围。合理的方案是交叉编址，即把连续地址的几个主存字依次分配在不同的存储体中，因为程序运行的局部性特性已经表明，程序运行过程中，在短时间内读写地址相邻的主存字的概率更大。假定有 M 个存储体，每个存储体的容量为 L，则第 m 个存储体中存储的主存字的地址应为

$$m * j + i$$

其中

$$j=0,1,2,\cdots,L-1, \qquad i=0,1,2,\cdots,M-1$$

在这种编址方式中，地址寄存器送到主存储器的地址的低几位(例如对4个存储体的情形为低2位)，用于区分读写哪个存储体，其余高位部分送到每个存储体，用于区分读写每个存储体的哪一个存储字。图7.18给出了该方案的原理示意图。

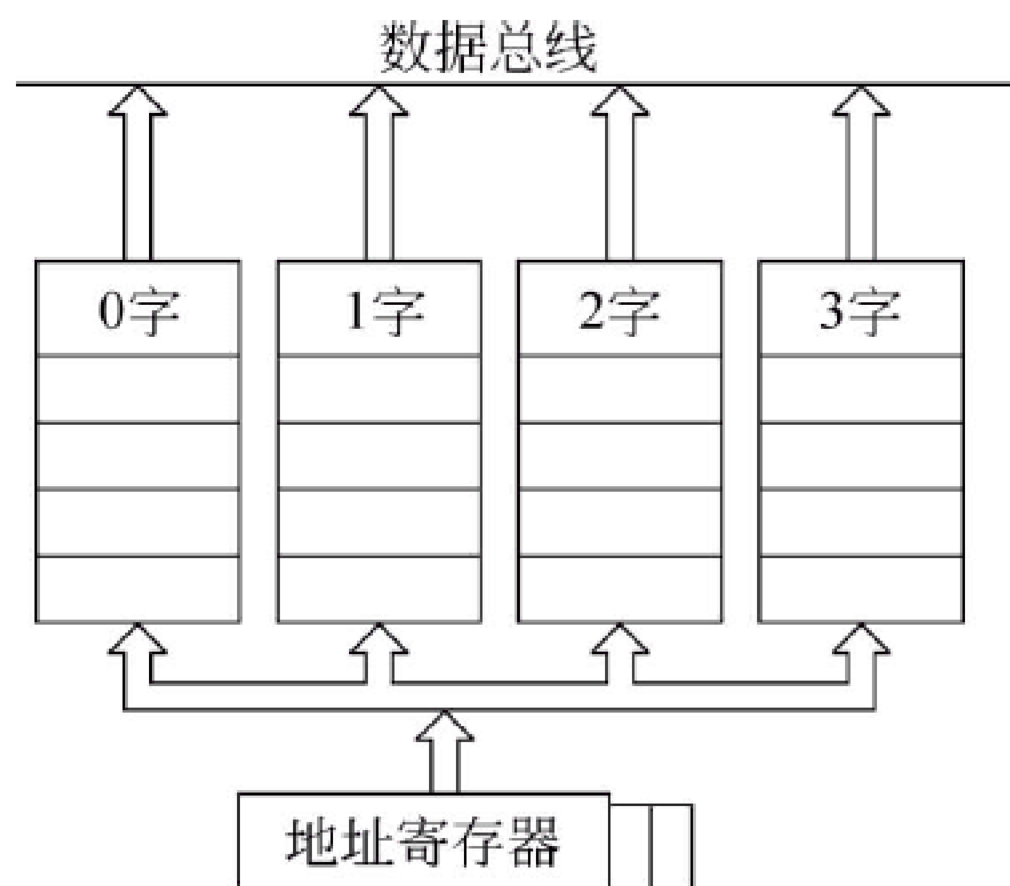


图7.18　主存的多体交叉编址技术

3. 关于对成组数据传送的支持

成组传送数据的方式(Burst Mode)是指用于提高在数据总线上的数据输入输出能力的一种技术，即通过地址总线传送一次地址后，能连续在数据总线上传送多个(一组)数据，而不像正常总线工作方式(Normal Mode)那样，每传送一次数据，总要用两段时间，即先送一次地址(地址时间)，后跟一次数据传送(数据时间)。在成组传送方式中，为传送 N 个数据，就可以仅用 $N+1$ 个总线时钟周期，而不再是用 $2\times N$ 个总线时钟周期，使总线上的数据I/O尖峰值提高一倍。

显然，实现数据成组传送，不仅是计算机总线一个方面的问题，也与提供数据来源(读)或数据去处(写)的主存储器直接有关。换言之，实现数据成组传送，CPU就要支持这种运行方式(486以上型号的PC才有此项支持)。主存储器也应能提供出足够高的数据读写速度，这往往通过主存的多体结构、动态存储器的EDO支持等措施来实现。这种支持可以在PC的内存条一级体现，也可能在存储器的芯片一级就有所体现。

4. 其他可行方案

存储器系统的性能主要表现为读写速度和存储容量两个方面。从进一步提高主存储器

部件的读写速度考虑，可以从提高存储器芯片本身的读写速度，改进芯片之间的组合与结构关系两个角度来加以解决。例如，使用增强型的动态存储器芯片 EDRAM、CDRAM 等方案，使用同步动态存储器芯片 SDRAM，改进芯片之间的组合与结构关系，或者选用具有多端口的存储器芯片等，都能达到更好的性能。具体内容请参阅其他资料，这里从略。

本章内容小结和学习方法建议

本章中对为什么要使用多级结构来构建存储器系统进行了说明。并且解释了程序运行的局部性原理和多级结构存储器系统中需要满足的一致性原则和包含性原则，这是学习第7～9章的概述性内容。

主存储器，又称内存储器，其容量与读写速度等指标对计算机总体性能有着重大影响。当前多用动态存储器芯片实现，它的运行速度比 CPU 要慢得多，其容量往往难以满足系统使用的要求，需要到存储器系统中找到解决方案。以教学计算机的存储器为例，介绍了存储器设计中的字、位扩展技术，存储器与 CPU 的连接关系等内容，这是本章的重点教学内容。有关存储器芯片的内部结构和读写原理等，大体属于线路方面的知识，可以适当地少花一点精力。提高主存器性能的思路和解决方案还是应该适当了解一些。

习题与思考题

1. 在计算机中，为什么要采用多级结构的存储器系统？它的应用是建立在程序的什么特性之上的？

2. 多级结构的存储器是由哪3级存储器组成的？每一级存储器使用什么类型的存储介质？这些介质的主要特性是什么？在多级结构的存储器系统中，何谓信息的一致性原则和包含性原则？

3. 比较 DRAM 和 SRAM 芯片的主要特性。

4. 衡量主存储器的性能主要看哪几项指标？应该如何确定主存储器的字长？

5. 为什么在一些大型的计算机系统中，通常总要使主存储器可以按字节(8位)、半字(16位)、字(32位)甚至于双字(64位)来寻址呢？

6. 在主存储器系统中，ROM 区的主要作用是什么？是不是计算机系统中都应有 ROM 存储区呢？

7. 为什么当前的计算机系统中，多选用 DRAM 芯片组成主存储器？你能解释为什么在教学计算机系统中要用 SRAM 组成主存储器吗？

8. 为什么动态存储器会是破坏性读出？为什么静态存储器读出操作不会破坏已存储的信息呢？

9. 动态存储器为什么要定期刷新？为什么刷新操作能够以行为单位进行？为什么不以字为单位执行刷新操作呢？有哪两种最常用的刷新方式？它们各自的优缺点是什么？刷新操作对存储器性能有影响吗？

10. 存储器读写操作时，地址信号、片选信号、读写命令、读出的数据或写入的数据，在时间配合上要满足什么关系？

11. 在所用主存储器芯片已确定的情况下,还要进一步大幅度提高主存储器系统的读写速度的办法是什么?

12. 主存一体多字和多体交叉方案的优缺点各表现在什么地方?低位地址的多体交叉是何含义?优点何在?

13. 下列各类存储器中,不采用随机存取方式的是________。

A. EPROM　　B. CDROM　　C. DRAM　　D. SRAM

14. 下列有关RAM和ROM的叙述中,正确的是________。

Ⅰ. RAM是易失性存储器,ROM是非易失性存储器

Ⅱ. RAM和ROM都采用随机存取方式进行信息访问

Ⅲ. RAM和ROM都可用作Cache

Ⅳ. RAM和ROM都需要进行刷新

A. 仅Ⅰ和Ⅱ　　B. 仅Ⅱ和Ⅲ　　C. 仅Ⅰ、Ⅱ、Ⅲ　　D. 仅Ⅱ、Ⅲ、Ⅳ

15. 某计算机存储器按字节编址,主存地址空间大小为64MB,现用4M×8位的RAM芯片组成32MB的主存器,则存储器地址寄存器MAR的位数至少是________。

A. 22位　　B. 23位　　C. 25位　　D. 26位

16. 某计算机主存容量为64KB,其中ROM区为4KB,其余为RAM区,按字节编址。先要用2K×8位的ROM芯片和4K×4位的RAM芯片来设计该存储器,则需要上述规格的ROM芯片数和RAM芯片数分别是________。

A. 1、15　　B. 2、15　　C. 1、30　　D. 2、30

17. 假定用若干个2K×4位芯片组成一个8K×8位的存储器,则地址0B1FH所在芯片的最小地址是________。

A. 0000H　　B. 0600H　　C. 0700H　　D. 0800H

第8章 高速缓冲存储器和虚拟存储器

在高速缓冲存储器一节，主要介绍高速缓冲存储器的构成、功能、读写原理、提高命中率的可行方案等内容。通常情况下，高速缓冲存储器对用户来说是透明的，用户无须了解它是如何运行的，甚至于不知道它是否存在，照样能设计并运行自己的程序。但是，高速缓冲存储器是计算机硬件的重要组成部分之一，还是应该了解它的构成方式（3种基本映像）和读写原理（以关联存储器方式存储和读出数据）等基本知识。

在虚拟存储器一节，介绍过虚拟存储器的概念和功能之后，重点讲解了段式和页式存储管理2种方案的硬件组成，从逻辑地址到内存地址的转换过程，属于入门性的知识。

8.1 高速缓冲存储器

高速缓冲存储器（Cache）是一个相对于主存来说容量很小、速度特快、用静态存储器器件实现的存储器系统。它的作用在于缓解主存速度慢、跟不上CPU读写速度要求的矛盾。它的实现原理是把CPU最近最可能用到的少量信息（数据或指令）从主存复制到Cache中，当CPU下次再用这些信息时，就不必访问慢速的主存，而直接从快速的Cache中得到，从而提高了得到这些信息的速度，使CPU有更高的运行效率。

这里讨论的是在“缓存-主存”层次遇到的问题。问题的焦点就集中到到底能有多大的概率，CPU可以从Cache中得到原本应该到主存中去取得的信息，这是评价Cache运行性能的关键指标，被称为Cache的命中率。造成这一问题的根源，是Cache的容量远远小于主存，它所存放的内容只是主存内容的很小一部分。为此，必须找出一整套方案，从Cache的读写原理，Cache的容量设置，确定Cache存储单元与主存哪一个单元是对应关系，主存和Cache每次交换数据的单位量、交换的时刻，Cache接入计算机系统中的方式等多方面，来解决Cache速度、命中率等一系列问题。这些正是在本节要讨论的主要内容。

8.1.1 Cache的运行原理

高速缓冲存储器的运行原理与主存储器的运行原理有很大区别。主存储器运行原理是建立在**每个主存地址对应主存的一个存储单元**这一关系之上的。在计算机程序中，要使用主存某单元中的数据，必须在指令中给出该单元的地址。读操作时，给出这一地址后，通过译码电路，就可以选中主存中被读的一个存储单元，执行读操作，读出的信息就是需要的数

据。而高速缓冲存储器的运行原理则完全不同,由于其存储容量很小,无法通过对原本用于读主存的地址直接进行译码来选择一个Cache单元。

找到一个主存单元所对应的Cache单元最简单的办法是合理设计Cache存储器的组织形式。例如,使Cache的每个存储单元由3部分内容组成。第一部分内容是Cache的数据字段,保存从主存某一单元复制过来的数据内容,这是在CPU第一次读出该主存字时完成的,在把读出内容传送到CPU的同时,顺便将该内容写进一个选中的Cache单元的数据字段。第二部分内容是Cache的标志字段,保存相应主存单元的地址信息,是在复制主存单元的数据内容的同时,把该主存单元的地址写进这一标志字段的,用它指明该Cache单元的数据字段部分保存的数据是从哪一个主存单元复制过来的。当程序中的一条指令要用一个主存地址读主存的某一单元时,就用这一地址来与Cache中的各个标志字段的内容相比较,若找到某一标志字段的内容与该地址值相同,则同一Cache单元的数据字段中的数据内容可能就是被读的数据;这表明,读Cache时,是用Cache单元内容的一部分(标志字段)的值(原本用于读主存的一个地址),来确定该Cache单元另外一部分(数据字段)的内容是否就是要读的数据。**以这种原理运行的存储器被称为关联(联想)存储器**。第三部分内容是Cache单元的有效位字段,规定其值为1,表示该Cache单元中的标志字段、数据字段的内容是有效的,为0,则说明该Cache单元在此之前尚未使用,其标志字段、数据字段的内容是无效的。有效位字段的内容,应在Cache刚投入使用时,清每个单元的有效位为0;在一个Cache单元被选中,且数据字段、地址字段的内容完成写入操作之后,该单元的有效位亦被置为1,表明该单元已被占用,其标志字段、数据字段的内容是有效的。

图8.1给出了Cache存储器最简单的读出原理,现在不妨先假定该Cache的一个存储单元与一个主存字相对应。Cache存储器要正常进行工作,当然还要有自己的管理逻辑部分,这一部分内容到后面有关部分再介绍。

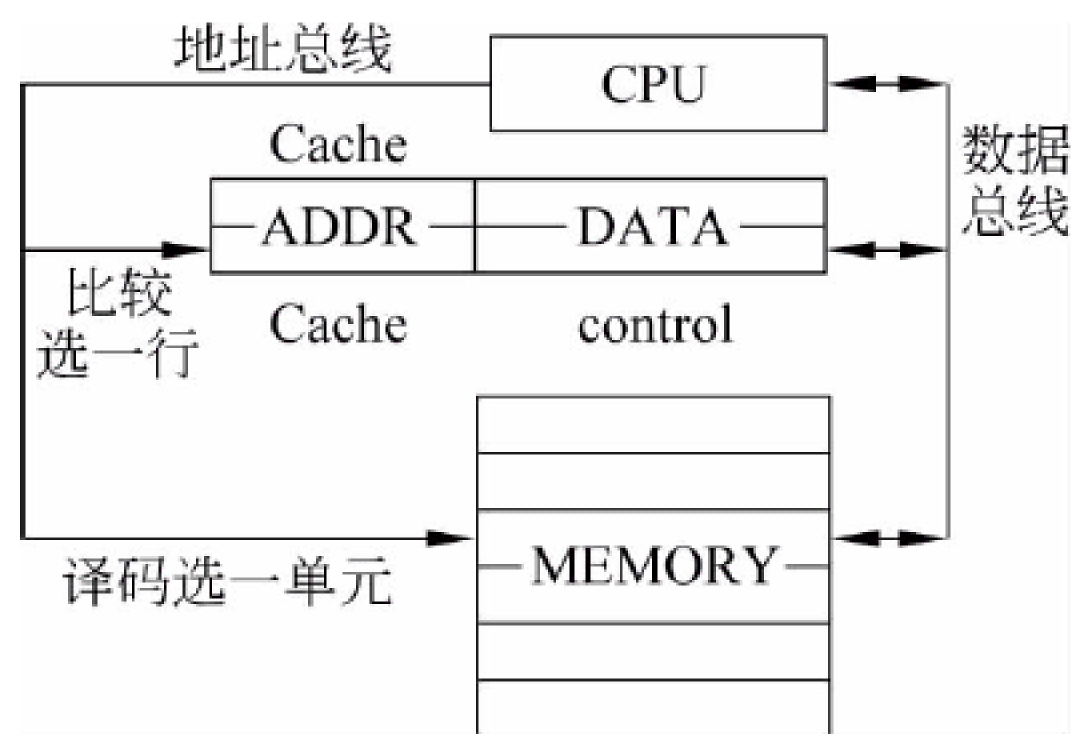


图8.1 Cache存储器读出简单原理

当然,实用的Cache存储器总会在简单原理的基础上采取一定的完善措施,以便尽可能地提高Cache性能,降低实现成本。这些完善措施包括以下4个方面。

(1) Cache单元不一定非要以字为单位与主存实现相互对应,因为以一个字为对应单位,Cache单元中的标志字段的长度必须能存放一个主存单元的完整的字地址,占用的位数偏多,使Cache的总容量变大,而且实现主存地址与Cache的标志字段比较的数量也变多(与Cache中所有有效位为1的存储单元都比较到),不利于提高Cache的运行性能和降低其实现成本。更常用的方案是以几个字组成的字块(Cache Line Size)为单位实现对应关系。

(2) Cache与主存交换信息时,也不是每次以一个主存字为单位进行交换,如果CPU在读一个主存字时,顺便把该字之后的几个字也写进Cache,则CPU再用到这几个字时,由于这几个字已在Cache中,CPU就可以快速从Cache中得到,而不必每次去读速度更慢的主存储器,从而提高了计算机系统的运行效率。更常用的方案是每次以几个字组成的字块为单位实现二者之间的信息传送,字块大小一般为4～32个字节。

(3) Cache单元与主存单元的对应关系,也可以有多种不同的方式,例如完全随意对

应，一对多的硬性对应，多对多的有限随意对应等。

（4）Cache 的容量设置、Cache 的分层组织、Cache 接入计算机系统中的方式等，也是设计与使用 Cache 过程中必须考虑的问题。

这里讨论的还只限于对 Cache 的读操作，对 Cache 写操作还会有其他需要解决的问题。下面会分别详细讲解。

8.1.2 Cache 的 3 种映像方式

前面已讲到，在把一个主存单元的数据复制到 Cache 中时，还要把该主存单元的地址，经过某种函数关系处理后写进 Cache 的标志字段，这一过程被称为 Cache 的地址映像。在程序执行时，还要把主存地址变换为访问 Cache 的地址，这一过程被叫作 Cache 的地址变换。地址映像和地址变换的处理方案是密切相关的。

Cache 存储器通常使用 3 种映像方式，它们是全相联映像方式、直接映像方式、多路组相联映像方式。3 种映像方式有各自的优缺点。

1. 全相联映像方式

全相联映像方式是指主存的一个字（字块）可以映像到整个 Cache 的任何一个字（字块）中，反过来说，Cache 的一个字（字块）中，在不同时刻可能存放的是整个主存中的任何一个字（字块）中的内容，即**二者的对应关系是完全随意的**，没有任何强制性的限制条件。这种方案的优点是明显的，对 Cache 的使用可以有最大的灵活性，只要 Cache 中尚有空闲的单元，又有新的主存单元的内容要写入 Cache 时，确保能执行这次写操作。当 Cache 已满，即所有单元都被占用，又有新的主存单元的内容要写入 Cache 时，可以比较方便地选择一个 Cache 单元进行腾空，接着完成这次写入操作。全相联映像方式缺点也是致命的，在执行 Cache 读写操作时，用原本读主存的整个（或部分）地址去与 Cache 的标志字段的内容实现比较时，必须与整个 Cache 中每一个单元的标志字段都比较到，才能知晓要读的信息是否已在 Cache 中。由于实现这一比较操作的电路过多过于复杂，其实现成本太高而难以实用，仅在 Cache 容量很小时方可选用。图 8.2 给出了这一方案的原理示意图。

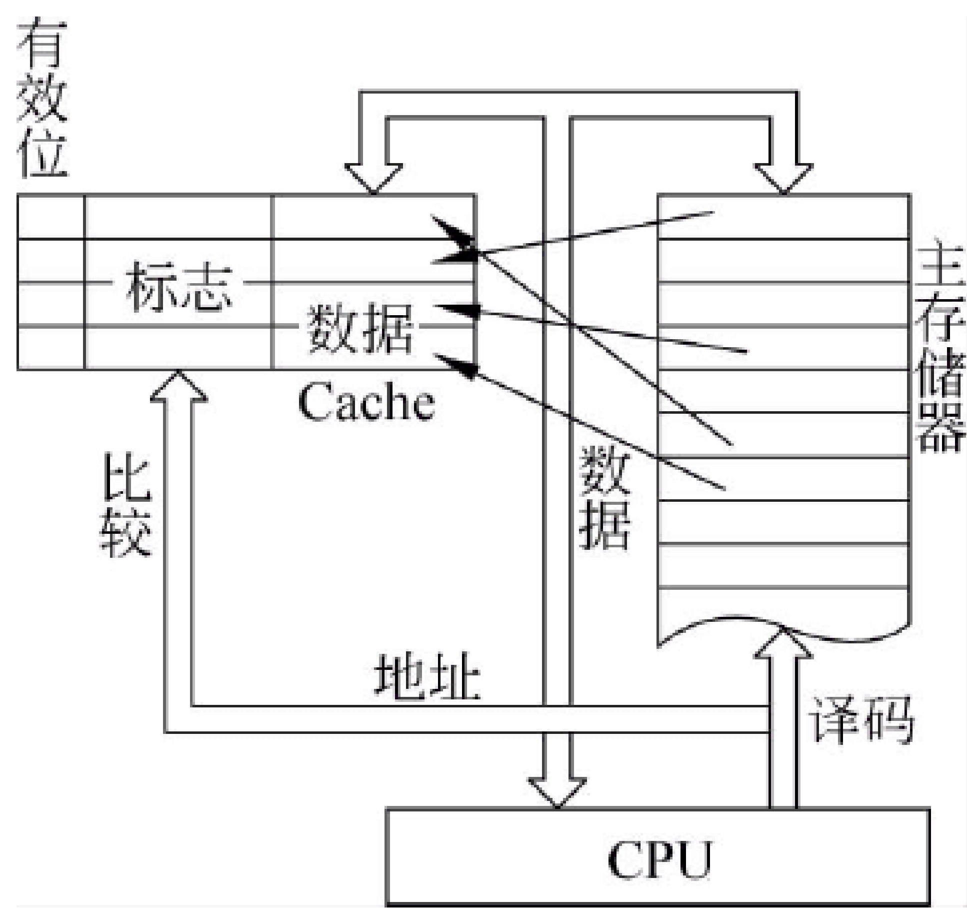


图 8.2　全相联映像方式

2. 直接映像方式

直接映像方式是指主存的一个字（字块）只能映像到 Cache 的一个唯一确定的字（字块）中，反过来说，Cache 的一个字（字块）中，在不同时刻存放的仅能是整个主存中确定的某些字（字块）中的一个字（字块）的内容，即二者的对应关系是完全硬性确定的，没有任何选择余地。这种方案的优点是实现与标志字段比较的线路简单，成本低；缺点是对 Cache 的使用缺乏灵活性，影响命中率。它的运行原理简述如下。

在逻辑上，把主存划分成容量等于 Cache 容量的许多“区段”，相应地把主存地址也划分为访问主存的“区段”（区段号）和区段内一个字（区段内偏移）这样两个字段，而在 Cache 的标志字段仅写入主存地址的区段号（地址映像）。在进行主存读写时，就可以用完整的主存地址访问主存的一个存储单元，并使用区段内偏移这一字段为地址（地址变换）访问 Cache

的一个单元,此时只须用主存地址的区段号的值与Cache的这一单元的标志字段的内容比较即可,二者相同且有效标志位的值为1,表明Cache的这一单元的数据字段的内容即为被读数据,即此次访问Cache成功,称为“命中”;二者不同时,则表明相应主存单元内容尚未读入Cache,是“缺失”,又称“不命中”。图8.3给出了这一方案的示意组织关系。

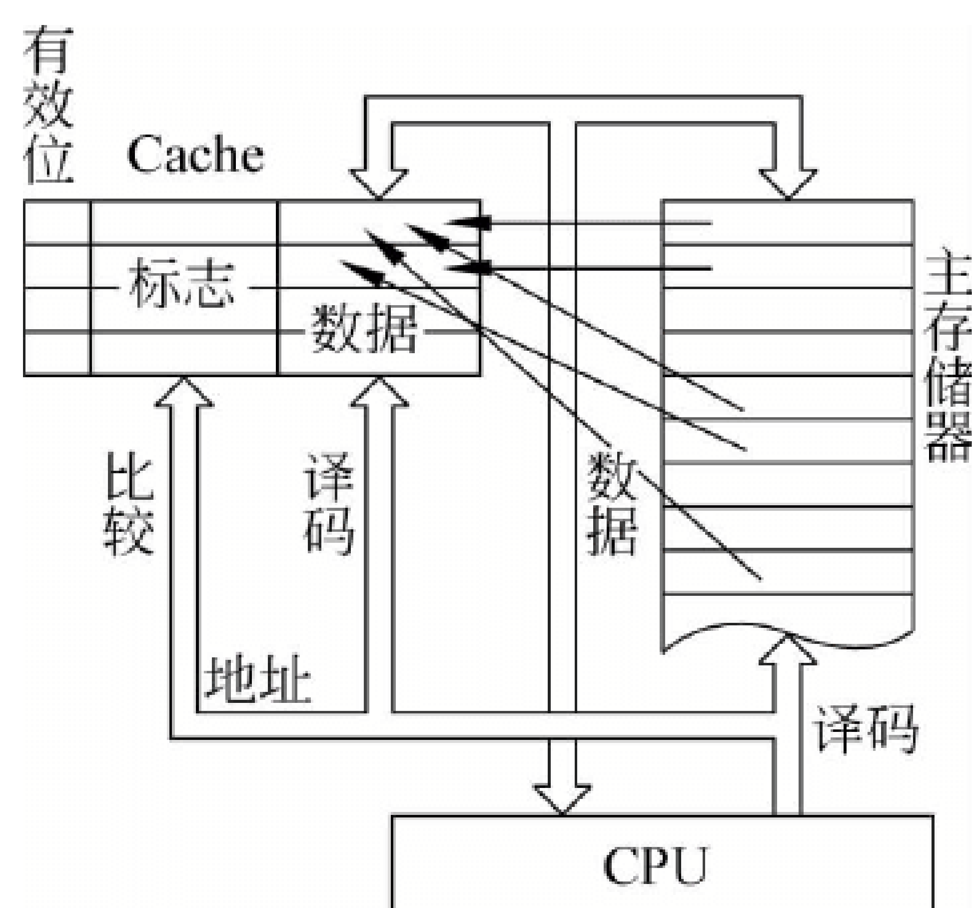


图8.3　直接映像方式

3. 多路组相联映像方式

多路组相联映像方式是对全相联映像和直接映像的一种折中的处理方案。它既不是在主存和Cache之间实现字块(字)的完全随意对应,也不是在主存和Cache之间的实现字块(字)的多对一的硬性对应,而是实现一种有限度的随意对应。例如,一个主存字块可以与Cache中的2、4、8个单元建立对应关系,此时被分别称为2、4、8路组相联。它的实现原理简述如下。

首先,把Cache存储器组织为同等容量的多体结构,例如变为2、4、8个体,若Cache的总容量不变,则每个体的存储单元数变为原来单体容量的若干分之一。每个Cache体的一个单元都可以和一个主存字(字块)的内容相对应,对应关系与直接映像类似,仍把主存划分成容量等于每个Cache体的存储单元数的多个“区段”,相应地把主存地址也划分为访问主存的“区段”(区段号)和区段内一个字块(区段内偏移)这样两个部分,而在Cache的标志字段仅保存主存地址的区段号。在进行主存读写时,就可以用完整的主存地址访问主存一个存储单元,并使用主存地址的区段内偏移这一字段为地址访问每个Cache体的一个单元,此时只需用主存地址的区段号的值同时与每个Cache体选中的一个单元的标志字段的内容比较,若与其中任何一个Cache体的标志字段的内容相同且相应有效位的值为1,则表明存储于这一Cache体中的这一数据字段的内容即为被读数据,即此次访问Cache命中;都不同时,则表明相应主存单元内容尚未读入Cache,是不命中。

这一方案与直接映像方案的区别主要表现在,每一个主存字块可以从多个(例如2、4、8个,而不再是一个)体中选择其一完成写入Cache的操作,有了更大程度的选择余地,有利于提高Cache的命中率。当用主存地址的区段号与Cache中的标志字段比较时,只与每个体的确定的一个存储单元的标志字段相比较,线路上实现也不会太复杂,确保这一方案简便易行。

这一方案与全相联映像方案的类同之处主要表现在,在把一个主存字写进Cache时,它可以在Cache的多个(例如2、4、8)体中进行选择,就是说,一个主存字(字块)可以与多个Cache单元(由每个Cache存储体提供一个)建立随意的对应关系,故这是有限度地随意对应,其目的是尽量简化完成与Cache中标志字段相比较的线路。当用主存地址的区段号与Cache中的标志字段比较时,只与每个体的确定的一个存储单元(而不再是全部单元)的标志字段相比较,线路上实现也不会太复杂,确保这一方案简便易行。多路组相联映像是性能最好的一种Cache组织方式。这一方案如图8.4所示。

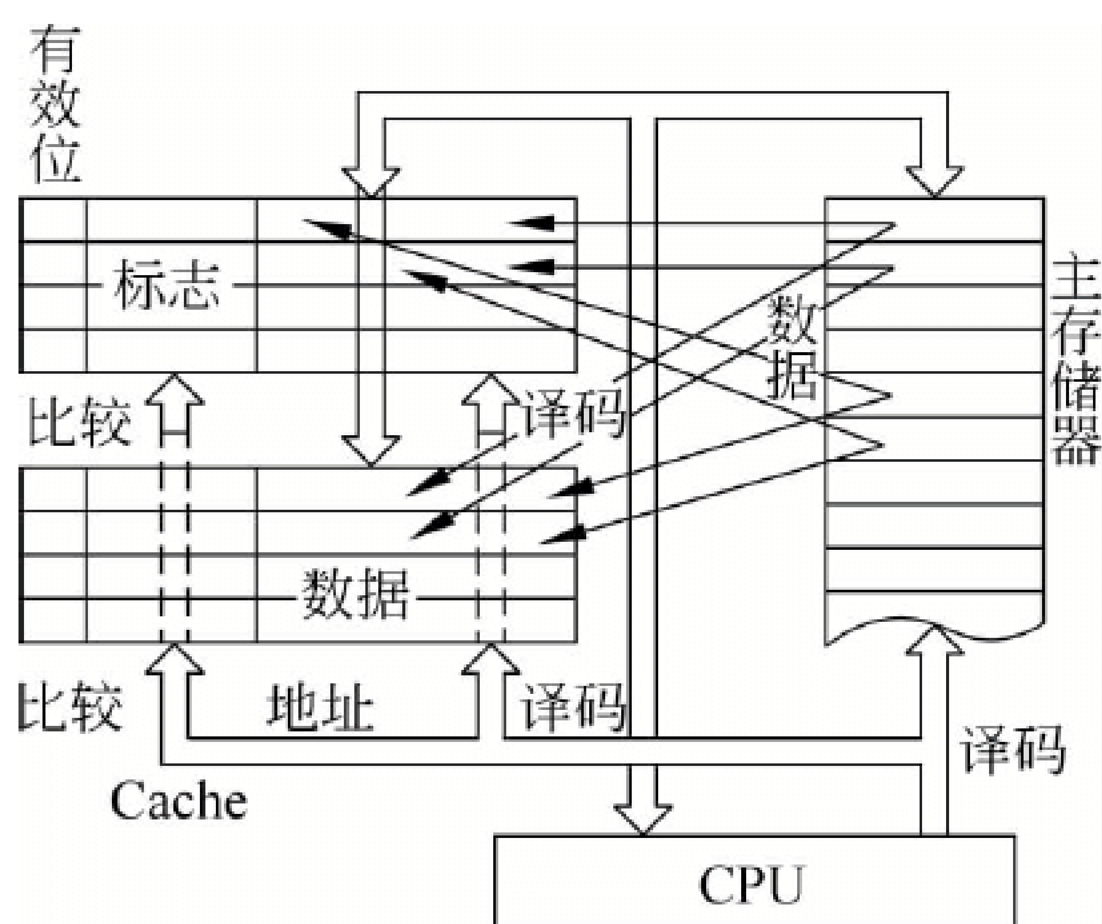


图8.4　二路组相联映像方式

设主存地址为 p 位，令 $p=m+b$，主存的字块总数为 2^m，块内字节数为 2^b。Cache 的地址码为 $q=c+b$ 位，Cache 的字块总数为 2^c，块内字节数与主存相同。令 j 为 Cache 字块号，i 为主存字块号。图 8.5 就 3 种映像方式主存地址的划分进行一个简单的比较，同时给出 3 种映像方式的地址映像和地址变换。

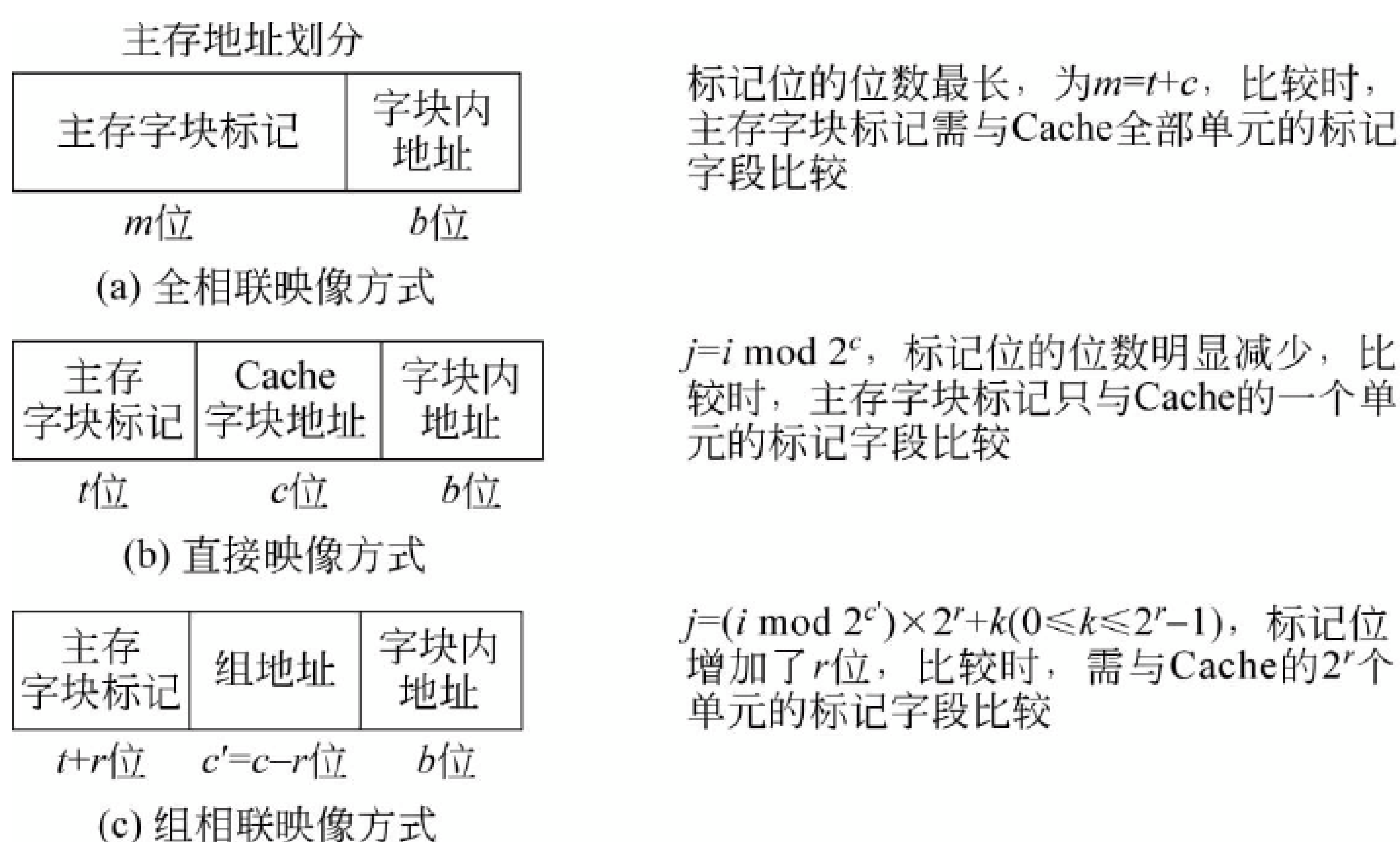


图 8.5 3 种映像方式的比较

8.1.3 Cache 实用中的问题

1. Cache 替换算法

Cache 中的块比主存中的块要少得多，前面我们介绍的 3 种映像方式，都表明一个 Cache 块可以对应多块主存块。因此，当新的一个主存块需要装入到 Cache 中时，就有一个将哪个 Cache 块替换出去的问题。对于直接映像方式，一个主存块仅和唯一的一个 Cache 块对应，也就不存在什么选择。而对于全相联映像方式和多路组相联映像方式，由于一个主存块可以装入到不同的 Cache 块中，就需要有一个替换算法来决定将哪一块 Cache 块替换出去。为保证 Cache 的性能，这个算法还应该用硬件实现。

迄今为止，已经使用了多个替换算法，其中效果最好的应该是最近最少使用法(Least Recently Used，LRU)，即替换出组内没有被访问时间最长的一个 Cache 块。对两路组相联方式，这个算法实现起来十分简单，只须在组内的两块 Cache 块中分别增加一个使用位，初始值均为 0。当某块中的内容被访问时，将它的使用位置为 1，而将另外的一块的使用位置为 0。这样，当有新的主存块要装入到这个组时，只需将使用位为 0 的那块替换出去即可。

除 LRU 法外，常用的算法还有先进先出法(First In First Out，FIFO)，即替换出最先装入到 Cache 中的那个主存块，这可以用一个循环电路来实现。还有一个算法是最少使用法(Least Frequently Used，LFU)，将使用次数最少的一个 Cache 块替换出去，它需要在每个 Cache 块上加一个计数器来记录 CPU 对其的访问次数。另外，也还有用随机替换算法随机找一块替换出去的。

2. 写 Cache 策略

为提高系统性能，CPU 主要与 Cache 交换数据，它对主存数据的更改，首先是更改 Cache 中的内容，为保持 Cache 中的数据与主存中数据的一致性，必然也要更改其对应的主

存中的数据。

现代计算机中,能够访问主存的设备已经不止一个。通常情况下,一个计算机系统内可能有多个CPU,它们可以有各自的Cache和公用的主存,也可能还有多个可以独立进行主存读写的设备(此时称它们为总线主设备(Bus Masters))。当任何一个CPU要完成写Cache/主存,或任何一个设备要执行读主存或写主存操作时,必须保证共享这些有关主存单元的CPU和总线主设备所用到的数据都是合法的(数据一致性原则)。

例如,一个外设向主存写入了一个数据,而该主存单元的内容在此之前已有一个副本在CPU的Cache中,此时,该主存单元内容与它在Cache中的副本就出现了不一致,怎么办呢?此时最简便的办法,就是把相应Cache单元中的有效位清掉,当CPU再次需要读这一主存单元时,它只能从主存中重新取得被外设修改过的新值,而不会使用原Cache中过时的旧值。又如,当CPU向自己的一个Cache单元中写入一个新值,但尚未修改与之相应的主存单元的内容时,也出现了二者间数据不一致性的问题,该怎样处理呢?常用的策略有写直达和拖后写两种。

第一种策略是写完Cache之后,立即修改主存,称为写直达(Write Through),其优点是处理简单,数据一致性容易保证。其缺点是,配置Cache对写操作来说,非但没有提高性能,甚至会降低系统的速度。更糟的情况也许会发生,在没有其他CPU、外设去读这些被重写过的主存单元时,这些写主存的操作是徒劳无功的,它只起到了减慢系统速度的作用。

第二种策略是写完Cache之后,不直接去改写相应的主存单元的内容,但把这些主存单元的地址登记到Cache控制器中的一张列表中,若有另外的CPU或外设发出读主存的请求,并将主存地址送到地址总线之上时,首先将它们要读的地址与原来保存在Cache的那张列表中的全部地址内容(可能多条)相比较,这一操作称为总线监听。若比较的结果都不同,表明本次的读请求使用的不是那些尚待修改内容的主存单元,则只需继续完成本次读操作即可;若与列表中的某一个地址相同,则表明本次读的主存单元目前保存的是一个无效(等待修改)的数据,不能允许读操作进行,需要首先发出命令停止该读操作,再把保存在Cache单元中的正确数据,通过一个写主存周期将其写入相应的主存单元,接下来再启动刚停下来的读操作,则CPU或外设读得的一定是正确的数据。这一策略的优点是明显的,它确保每一次写主存都是必须完成的那一部分,而不全出现徒劳的写操作,系统性能高。这种策略称为拖后写(Write Back)。其缺点是实现上比较复杂,成本较高。

3. Cache的命中率

Cache的命中率是Cache最重要的属性,提高Cache的命中率是存储器系统设计者的目标。影响Cache命中率的因素有很多,包括Cache的容量、Cache块的大小、Cache的组织方式及Cache的数量等。

一般来说,因为可以同时在Cache中存放有更多的内容,增加Cache的容量显然可以提高Cache的命中率,命中率与Cache容量的关系可以用图8.6来表示。可以看出,随着Cache容量增大到一定的程度,再增加容量对提高命中率的效果就不大了。而由于组成Cache必须是价格昂贵的SRAM,所以,从性能/价格比考虑,Cache容量一般应控制在合理的范围内。

Cache与主存交换数据是以块为单位的,当CPU要装入一个主存字到Cache内时,不仅仅是装入它需要的这一个字,而是包括与之相邻的其他字的整个主存块。当Cache块的

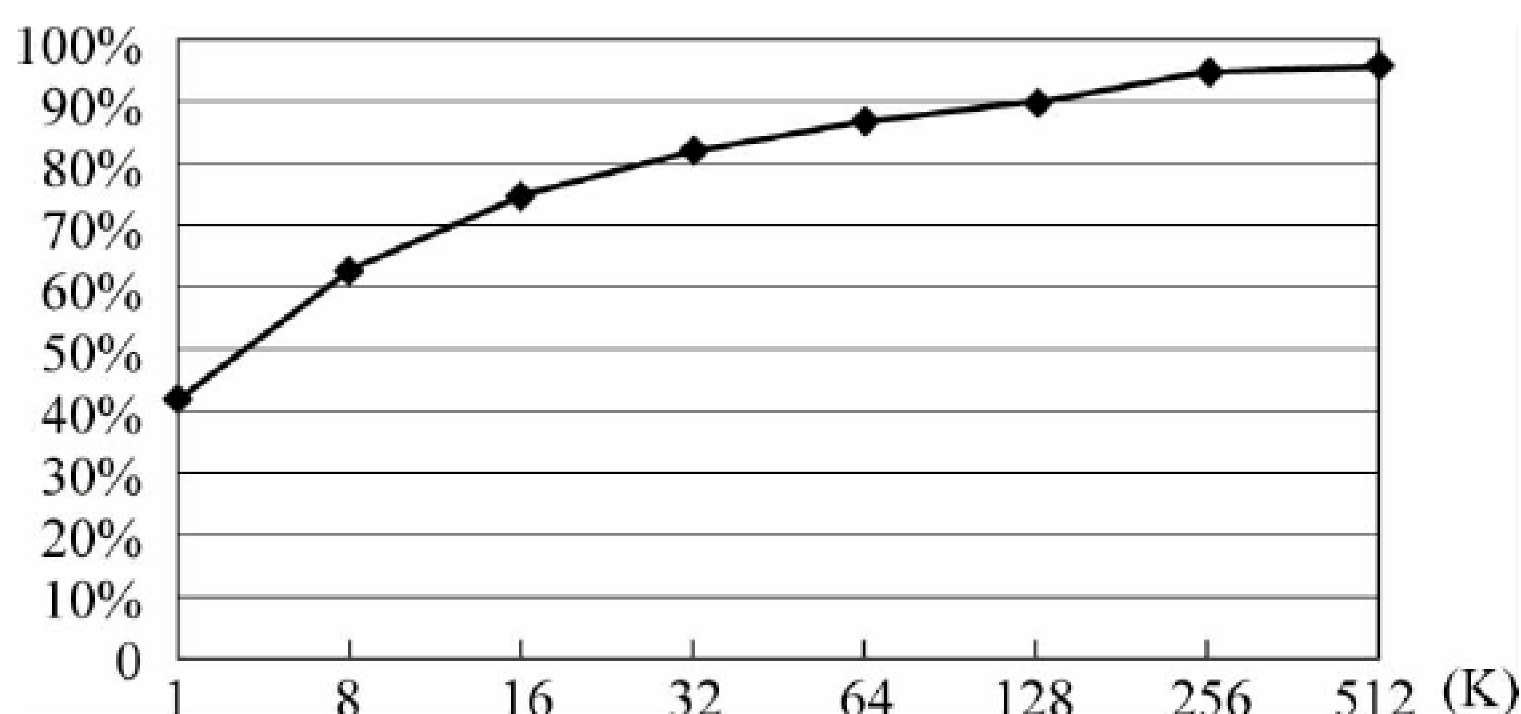


图 8.6　Cache 命中率和容量的关系

大小增加时,每次交换的数据量增加,根据程序局部性原理,命中率应该会增加。当然,Cache 块的大小也不是越大越好,大的 Cache 块每次和主存进行数据交换时需要的时间也长,而且,装入 Cache 的内容也许是 CPU 近期不需要的数据,可能会造成不必要的数据装入,影响系统的性能。一般来说,Cache 块的容量在 4～32 个字节范围之间比较适宜。

Cache 的组织结构也会影响 Cache 的命中率。显然,直接映像方式的命中率会比较低,而全相联映像方式的命中率比较高。但从性能/价格比综合考虑,最合适的组织方式应该是两路或四路组相联方式。

Cache 刚出现时,系统中一般只有一个 Cache。目前,多 Cache 的计算机系统已经很普遍了。通常的方法是在已有的 Cache 存储器系统之外,再增加一个容量更大的 Cache。此时原有 Cache 为第一级 Cache(例如奔腾微处理机芯片内的 Cache),新增加的 Cache 则成为第二级的 Cache。这两级 Cache 的关系中,第二级 Cache 的容量比第一级 Cache 的容量要大得多,在第一级 Cache 中保存的信息也一定保存在第二级 Cache 中(多级存储器系统中的包含性原则),但保存有比第一级 Cache 中更多的信息。当 CPU 访问第一级 Cache 出现缺失情况时,就去访问第二级 Cache。依此思路,也可以再增加第三级 Cache。若第一级、第二级 Cache 的命中率为 90%,则它们合起来之后的命中率将为 1－(1－90%)×(1－90%)＝99%,而不会是 81%。

4. Cache 的接入法

如何把 Cache 接入计算机的系统,也是值得认真解决的问题,常用的接入 Cache 的方法有以下两种,如图 8.7 所示。

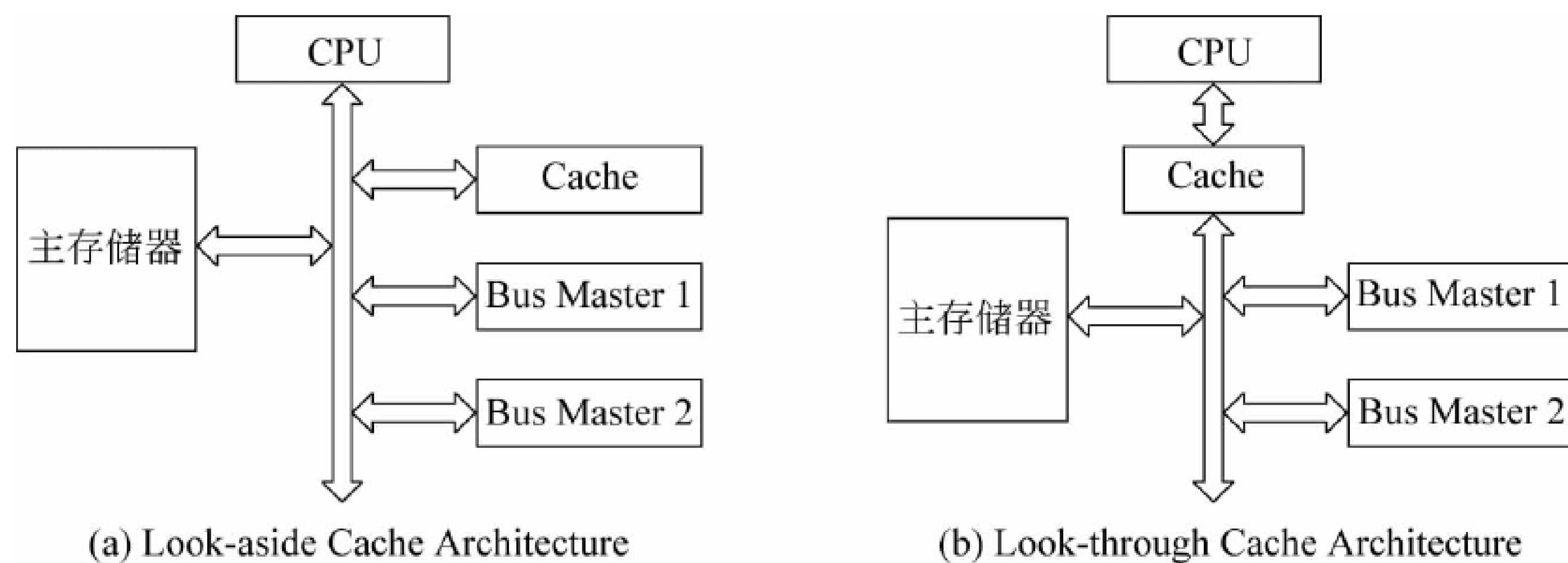


图 8.7　Cache 接入系统的方法

第一种方法是把 Cache 系统如同一个接口卡一样的接到计算机主总线上,称为 Look-aside Cache Architecture,如图 8.6(a)所示。这一结构的优点是实现简单、成本低;其缺点

是,当 CPU 与 Cache 交换信息时,它占用了总线,使接在总线上的其他 Bus Masters 不能与主存交换信息。

第二种方法是把原来的处理机总线“打断”,用 Cache 把原来的总线隔断为两个部分,称为 Look-through Cache Architecture,如图 8.6(b)所示。它的优点是提高了总线上并发操作的可能性,即当 CPU 与 Cache 交换信息时,Bus Master 可以同时与主存交换信息,有利于提高计算机整体性能。其缺点是设计复杂,实现成本较高,而且在读 Cache 不命中的情况下,读主存用的时间会更多一点,它在读 Cache 失败之后才启动读主存。

8.2 虚拟存储器

8.2.1 虚拟存储器的概念介绍

虚拟存储器通常是指高速磁盘上的一片存储空间,其功能是通过硬件、软件的办法,可以将其作为主存储器的扩展的存储空间一样来使用,这就使得程序设计人员能够使用比主存储器实际容量大得多的存储空间来设计和运行程序。这里讨论的是在“主存—辅存”层次遇到的问题。众所周知,主存储器的容量一直是影响计算机性能的关键因素,人们又不想通过花太多的钱,无节制地扩大主存储器实际容量,出路何在呢? 根据程序运行的局部性原理,一个程序运行时,在一小段时间内,只会用到程序和数据的一小部分,仅把这部分程序和数据装入主存储器即可,更多的部分可以在用到时随时从磁盘调入主存储器,这就是提出虚拟存储器的核心思路。

虚拟存储器所追求的目标是摆脱主存储器容量的限制(通过磁盘非常大的存储空间解决),降低存储一定信息所用的成本(通过磁盘非常低的存储成本解决)。问题是应该如何实现一个程序和相关数据在磁盘和主存储器之间交换;如何划分磁盘和主存储器的区域;如何管理(何时、何种策略、每次交换信息的单位量)二者之间的信息交换;要增加一些什么硬件和软件组成,这些就是我们将在本节讲解虚拟存储器过程中要讨论的问题。要指出,虚拟存储器的有关知识不是计算机组成原理课的重点内容,在计算机系统结构、操作系统两门课程中将有更详细、系统的说明。我们这里只简明扼要地介绍虚拟存储器中的段式存储管理和页式存储管理的基本概念和原理性知识。

8.2.2 段式存储管理

从程序设计和存储器管理的角度看,段这个词有特定的含义。段是模块化程序设计的产物,在程序设计过程中,通常会把在逻辑上、处理功能上有一定独立性的程序段落单独划分成一个独立的程序单位,供主程序或其他程序部分调用,一个大的程序是由许多程序单位经过连接组成的。此时的每个程序单位就是一个程序段,可以用段名或段号来指明程序段,每个段的长度是随意的,由组成程序段的指令条数决定,或由组成数据段的数据数目决定。在处理和运行这样的程序时,把段作为信息单位,实现在主存—辅存之间传送和定位是合理的。为此,必须把主存按段进行分配与管理,这种管理方式被称为段式存储管理。

经过连接而组成的程序所占用的空间被称为程序的逻辑空间。在指令中,一个逻辑地址由逻辑段号拼接上段内地址组成(每段内的第一个字的段内地址均默认为 0)。在程序运

行过程中，当用到某一段并将其调入主存时，它被分配在一片连续的主存区域，该主存区域的起始单元用于存放该段的第一个字，以后各字均依次顺序存放。图 8.8 给出了由 3 段组成的一个程序的逻辑地址空间，其中两段已装入主存的示意表示。

由此可见，段式存储管理的核心问题是变逻辑地址中的逻辑段号为主存中的一个存储区域的起始地址，这是通过在系统中设置一个段表完成的。段表也是一个特定的段，通常被保存在主存中。为访问段表，段表在主存中的起始地址被写入到一个被称为段表基地址寄存器的专用的寄存器中。段表由多个入口（表项）组成，每个表项由 3 部分内容构成：段起始地址、段的长度和段的装入位。段起始地址给出的是本段在主存中的起始地址，该起始地址加上段内地址就得到本段的一个字在主存中的真正地址。段的长度用于主存使用的合法性检查，当出现段内地址超过段的长度时，就是主存使用中的一个地址越界错误。段的装入位用于判断本段是否已装入主存。图 8.9 给出了由逻辑地址到主存实际地址的转换过程。

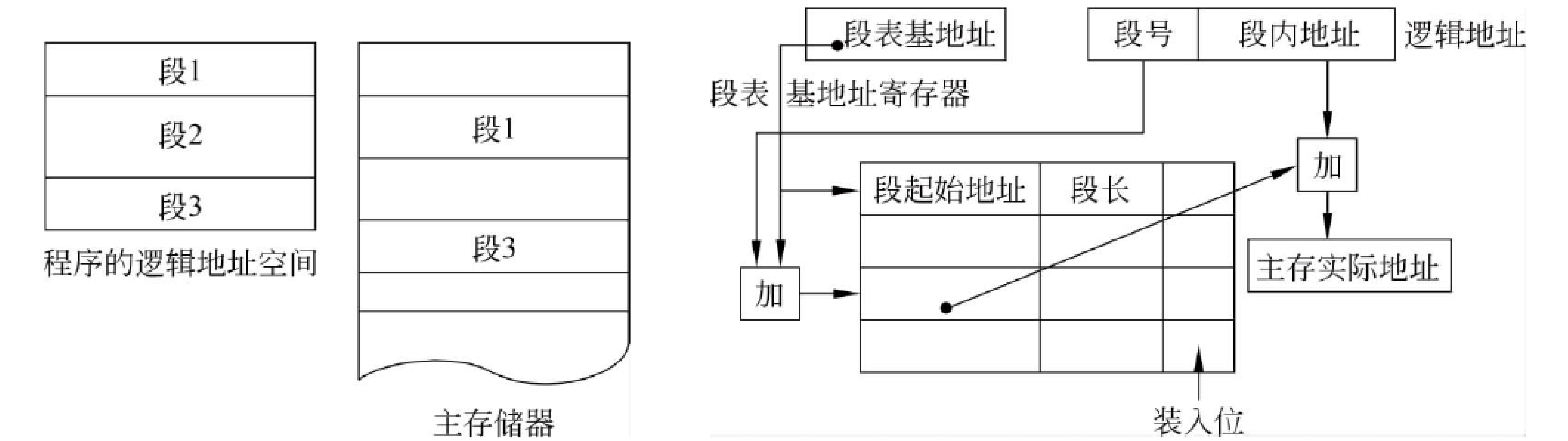


图 8.8　逻辑地址与主存占用示意表示

图 8.9　逻辑地址到主存实际地址的转换

地址转换过程从概念上讲可以用如下办法完成：把逻辑地址中的段号取来与段表基地址的内容相加，用相加之和做地址，找到段表的一个表项，检查该表项中的装入位，其内容为 1，表示该段已调入主存，从表项中取段的起始地址与逻辑地址中的段内地址相加，就得到一个数据在主存中的实际地址。若表项中的装入位的值为 0，表示该段尚未调入主存，则操作系统负责首先把该段从磁盘装入主存，并相应修改段表中的该表项内容，之后才可以完成地址转换过程。实际实现中，还会设立一个段地址寄存器，当一段刚投入运行过程时，完成前述操作之后，会把这一段在主存中的起始地址首先写入段地址寄存器，在该段其后的运行过程中，就不必再查段表，而用段地址寄存器的内容直接与段内地址相加，以便快速地得到一个数据在主存中的实际地址，从而提高系统的运行性能。

段式存储器管理的优点是明显的。首先，段的分界与程序的自然分界相对应。其次，段逻辑上的独立性使其易于分别编译、管理、修改和保护，也便于多道程序实现对段的共享。但段长的不确定性，会给主存空间的分配与管理带来麻烦，而且容易造成在段间留下许多零碎的、难以使用的小的存储空间（称为碎块），浪费存储器的有效可用容量，假若碎块太多，甚至会使整个存储器都难以利用。

8.2.3　页式存储管理

页式存储管理是另一种经常用到的虚拟存储器管理技术。它的主要思路是把虚拟（逻辑）地址空间和主存实际（物理）地址空间都分成容量大小相等的页，并规定页的大小为 2 的

整数次方个字,则所有地址都可以用页号拼接页内地址的形式来表示。虚拟地址用虚页号拼接页内地址给出,主存实际地址用实页号拼接页内地址给出。请注意,页式存储管理与段式存储管理的一个重要区别,段本身是程序设计的一个产物,段是一个独立的程序单位,长度可变;而页则不是程序本身的特性,是为了方便管理,人为地对程序进行划分的结果,通常在一个计算机系统中,页的长度是事先确定的,不会变化。这种通过分页方式进行的存储器管理被称为页式存储器管理。它的关键功能是实现以页为单位来完成在虚存和主存之间信息交换,并完成逻辑地址到物理地址的转换。说到底就是找出虚页号和实页号的对应关系。这可以通过设立页表来完成。

页表由若干表项组成,每个虚页号对应页表中的一个表项,表项的内容可以由如下一些部分组成。最重要的是一个虚页分配在主存中的实际页号,还可能包括页装入(有效)位,修改标记位,替换控制位,其他保护位等组成的控制位字段。请注意,在页式存储器管理的系统中,页表本身也是以页为单位管理的,可以保存在虚存中,也可以保存在主存中,为了能到主存中找到页表,必须设立一个专用的页表基地址寄存器。图8.10给出了页式存储器管理的地址变换过程。

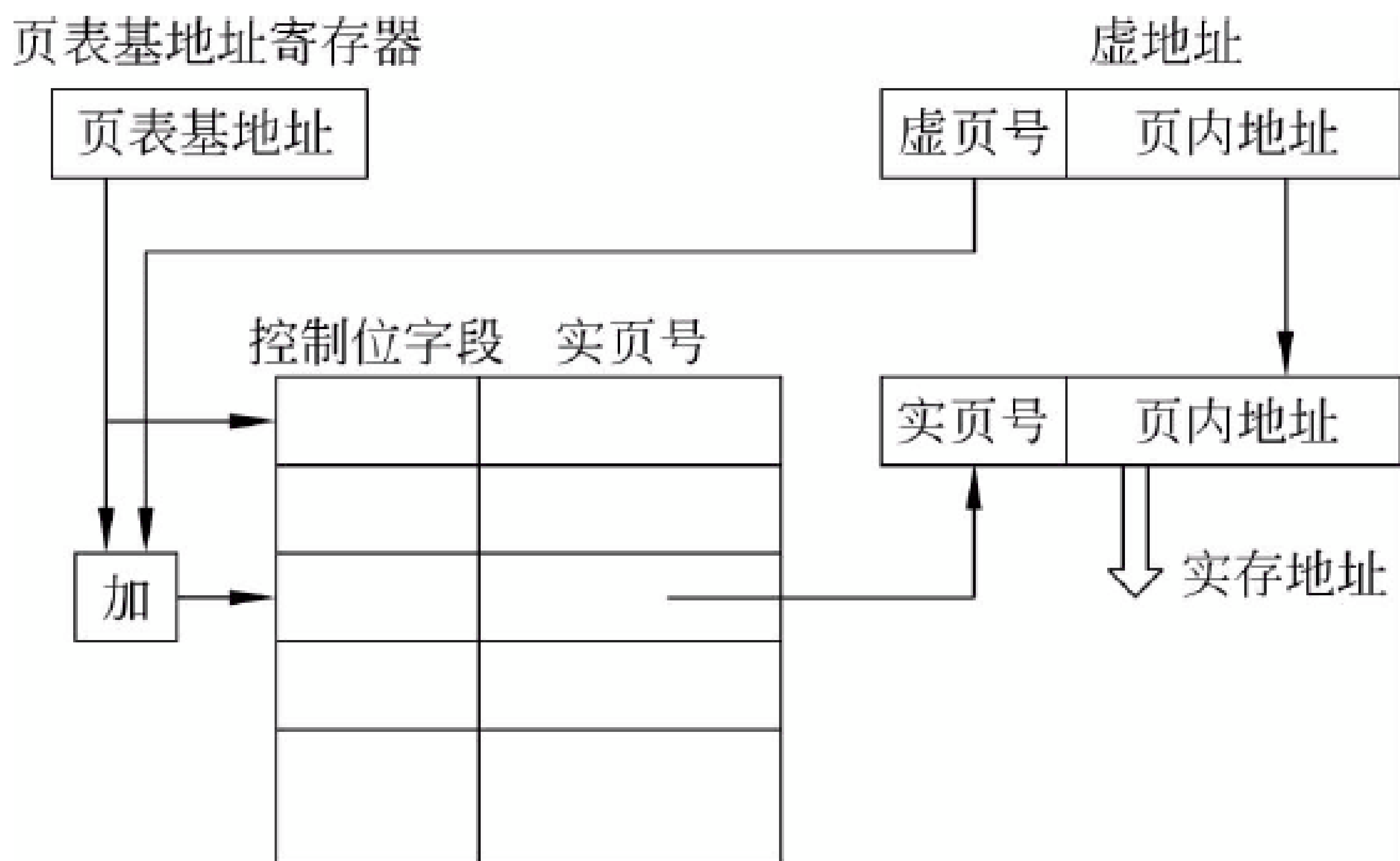


图8.10　页式存储器管理的地址变换过程

这一地址变换过程是:用虚地址中的虚页号与页表基地址相加,求出对应该虚页的页表表项在主存中的实际地址,从该表项的实页号字段取出实页号再拼上虚地址中的页内地址,就得到读主存数据用的实际地址。

当需要把一页从虚存内容调入主存时,操作系统从主存储器的空闲区找出一页分配给这一页,把该页的内容写入主存,把主存储器的实际页号写进页表的相应表项的实页号字段,写装入位为1。

当下次要读该页内的某个存储单元时,首先要读一次主存,通过查页表求出实存地址,然后再读一次主存,才能取得要读的数据,为读一个数据变成两次读主存,实际应用中是难以令人接受的。怎么解决这一问题呢?答案是设立一个完全用快速硬件实现的容量很小的(一般在16～64个表项之间)快速页表(英文为Translation Lookaside Buffer,TLB),用于存放在页表中使用最频繁的、为数不多的那些表项的内容。它的最重要的两项内容是虚页号和实页号,如图8.11所示。经快速页表实现的地址转换过程,用虚地址中的虚页号去与快速页表中虚页号字段的内容相比较,与哪个表项中的虚页号相同,则可以取出该表项中的实页号,并与页内地址拼接出主存实际地址。这一过程可以很快完成,很类似于高速缓冲存

储器的运行原理。当在快速页表中找不到该虚页号时，就要到主存中经慢表找出该虚页号对应的实页号，在得到一个主存实际地址的同时，并用该虚页号和实页号替换快速页表的一个表项的内容，以反映这次操作的现实形势。

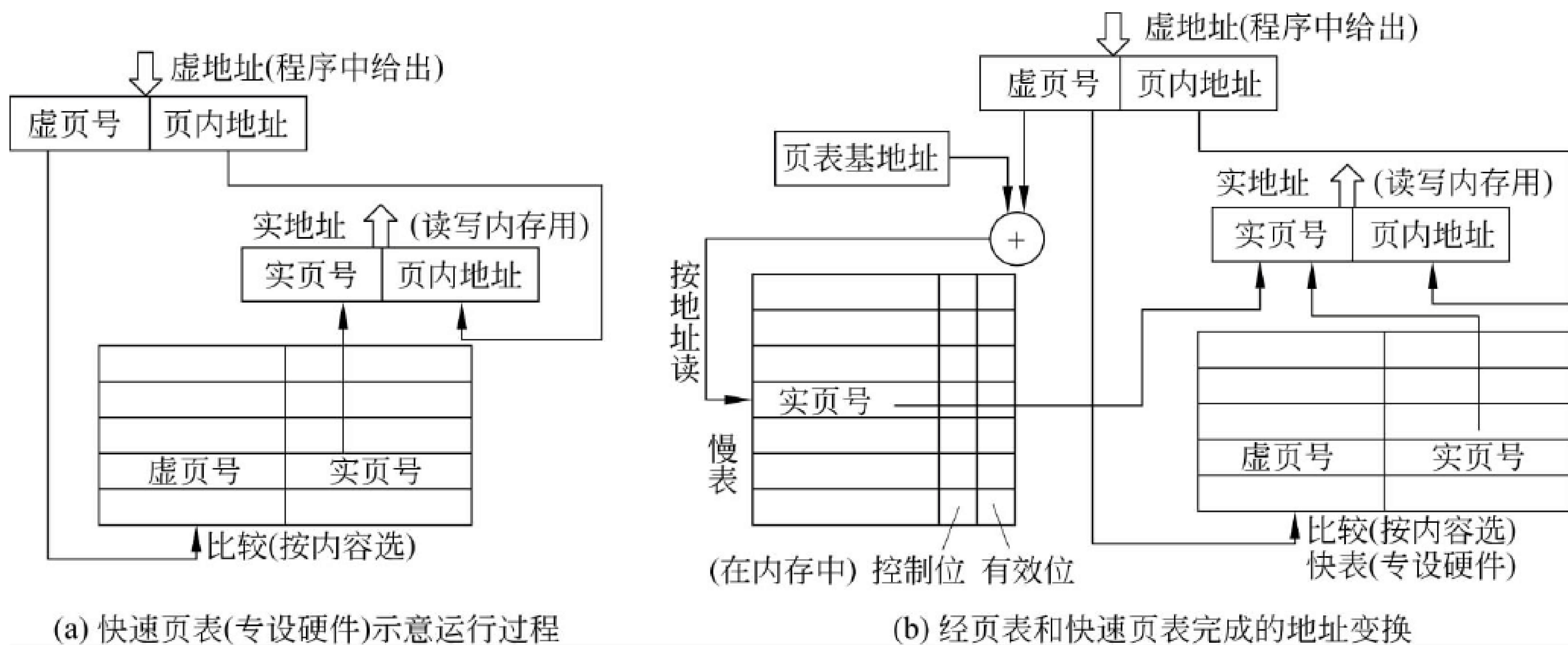


图 8.11　页式存储器管理中经快速页表实现的地址变换

本章内容小结和学习方法建议

在简单介绍高速缓冲存储器的功能和基本运行原理的基础上，重点讲解了 Cache 的全相联映像、直接映像和多路组相联映像 3 种构成方式，比较它们的优缺点，讨论了提高高速缓冲存储器命中率的一些措施。在介绍虚拟存储器的功能和概念的基础上，仅就段式和页式两种管理方案，强调虚拟存储器的硬件组成和把逻辑地址转换为内存实际地址的办法。本章内容强调基本原理和概念，没有特别关注具体例子和实用等方面的问题。

习题与思考题

1. 高速缓冲存储器在计算机系统中的主要作用是什么？用什么类型的存储器芯片实现？为什么？高速缓存与主存在读写原理方面有何区别？

2. 高速缓冲存储器有哪 3 种主要的映像方式？从地址映射和地址变换比较它们各自的特点。

3. 当主存或高速缓存某单元被改写后，通常是如何保证二者间的数据一致性原则的？不同方案的优缺点各是什么？从中是否进一步认识到在多级结构的存储器系统中，保证数据一致性原则的重要性？

4. 说明影响高速缓存命中率的因素都有哪一些，并简单解释一下是如何影响的。

5. 高速缓存与主存通常只以字为单位交换信息吗？为什么？若以几个字为单位交换，CPU、总线与主存应提供什么支持？

6. 使用多级高速缓存的目的是什么？

7. 虚拟存储器是个什么概念？它的存储介质是什么？虚拟存储器要解决的是什么

问题?

8. 虚拟存储器要求有什么硬件与软件支持?段式存储管理与页式存储管理的区别表现在哪些方面?段式存储为什么会在内存储器中形成"碎块"?

9. 说明段表的组成与逻辑段地址到内存实际地址的变换过程。

10. 说明页表的组成与程序逻辑地址到内存物理地址的变换过程。快速页表是一定要有的吗?说明快速页表内容的组成与读写原理。

11. 假设某计算机的存储器系统由 Cache 和主存组成。某程序执行过程中访存 1000 次,其中访问 Cache 缺失(未命中)50 次,则 Cache 的命中率是________。

A. 5% B. 9.5% C. 50% D. 95%

12. 某计算机的 Cache 共有 16 块,采用 2 路组相联映像方式(即每组两块),每个主存块大小为 32 字节,按字节寻址,主存 129 号单元所在主存块应装入到的 Cache 组号是________。

A. 0 B. 2 C. 4 D. 6

13. 下列命令组合情况中,一次访存过程中不可能发生的是________。

A. TLB 未命中,Cache 未命中,Page 未命中

B. TLB 未命中,Cache 命中,Page 命中

C. TLB 命中,Cache 未命中,Page 命中

D. TLB 命中,Cache 命中,Page 未命中

14. 已知某计算机系统共有 4KB Cache,采用多路组相联映像方式,分为 32 组,每组有 4 个 Cache 块。其地址长 32 位,最小编址单位为字节。

(1) 若不考虑用于 Cache 一致性维护和替换算法的控制位,则 Cache 的总容量为多少?

(2) 内存地址 000010AFH 将映射到 Cache 中的哪一组(从 0 开始编号)?

(3) 若内存地址 000010AFH 和 FFFF7xyBH 可以同时被映射到 Cache 中的同一组,那么 xy 可能的取值为多少?

15. 某计算机的主存地址空间大小为 256MB,按字节编址。指令 Cache 和数据 Cache 分离,均有 8 个 Cache 行,每个 Cache 行大小为 64B,数据 Cache 采用直接映像方式。现有两个功能相同的程序 A 和 B,其伪代码如下所示。

```
程序 A:
    int  a[256][256];
    ⋮
    int sum_array1()
    { int i,j,sum=0;
    for ( i=0;i<256;i++)
      for ( j=0;j<256;j++)
        sum+=a[i][j];
    return sum;
    }
```

```
程序 B:
    int  a[256][256];
    ⋮
    int sum_array2()
    { int i,j,sum=0;
    for ( j=0;j<256;j++)
      for ( i=0;i<256;i++)
        sum+=a[i][j];
    return sum;
    }
```

假定 int 类型数据用 32 位补码表示,程序编译时 i,j,sum 均分配在寄存器中,数组 a 按行优先方式存放,其首地址为 320(十进制数)。请回答下列问题,要求说明理由或给出计算

过程。

（1）若不考虑用于 Cache 一致性维护和替换算法的控制位，则数据 Cache 的总容量为多少？

（2）数组元素 a[0][31]和 a[1][1]各自所在的主存块对应的 Cache 行号分别是多少？（Cache 行号从 0 开始）

（3）程序 A 和 B 的数据访问命中率各是多少？哪个程序的执行时间更短？

第 9 章 外部存储器设备

本章主要介绍的是磁盘、光盘等外部存储器(简称外存)。当高速磁盘以虚拟存储器的面貌出现时,人们更喜欢称其为辅助存储器。外存设备更多的是强调它们的如下特性:设备的存储容量大、存储成本低,特别是在断电后仍能长期保存信息,大部分存储介质还能脱机保存信息。本章内容属于一般了解的部分比较多,例如磁表面存储器的读写原理,相关设备的一般组成等;属于应该掌握的内容不是很多,主要是磁记录编码,磁盘、光盘简单组成与读写原理,磁盘阵列与容错技术等概念性知识。

9.1 外存设备概述

9.1.1 主要技术指标

(1) 存储密度。在存储介质的单位长度上或单位面积上所存储的二进制信息的数量。对于磁盘设备,通常用道密度和位密度表示,也可以用二者的乘积表示。对磁带设备,通常总是用位密度表示。

(2) 存储容量。一台设备所能存储的总信息数量,通常以字节为单位表示。

(3) 寻址时间。磁盘设备属于按直接存取方式读写的设备,它的寻址时间由两部分组成,一是磁头沿磁盘的径向运动到目标磁道的时间;二是在目标磁道上等待磁盘被读写区段旋转到磁头下面的时间。由于这两个时间都与当时磁头距离目标位置的远近有关,故习惯上都用这两个时间各自的最大与最小时间的平均时间之和来表示。当前的磁盘,该值为几个毫秒到十几个毫秒;磁盘本身的读写速度相对较快。磁带设备是以顺序存取方式完成读写,不存在寻找磁道的问题,但要读写磁带某个区域上的信息,必须首先等待磁带旋转到该区域,其所用的时间,可能是几分钟到十几分钟,时间会比较长;磁带本身的读写速度也要慢一些。

(4) 数据传输率。指磁表面存储器在单位时间内可以向主机传送数据的数量,通常用二进制数的位数或字节数表示。它与设备本身的读写速度和接口逻辑线路有关。

(5) 误码率。外存设备是高精密度的机械电子装置,集机械、电子、电磁、光电等多项技术于一身,不仅价格较高,其运行的可靠性也远比 CPU、主存等电子逻辑部件要低。误码率就是用于衡量磁表面设备运行可靠性的重要指标,它等于在一次读操作过程中,出错信息数量在读出的全部信息中所占的比例。

9.1.2 磁记录原理与记录方式

1. 磁记录原理

磁表面记录设备是在磁头和磁性材料的记录介质之间有相对运动时，通过一个电磁转换过程完成读写操作的。图9.1给出了磁头结构和电磁转换的示意图。

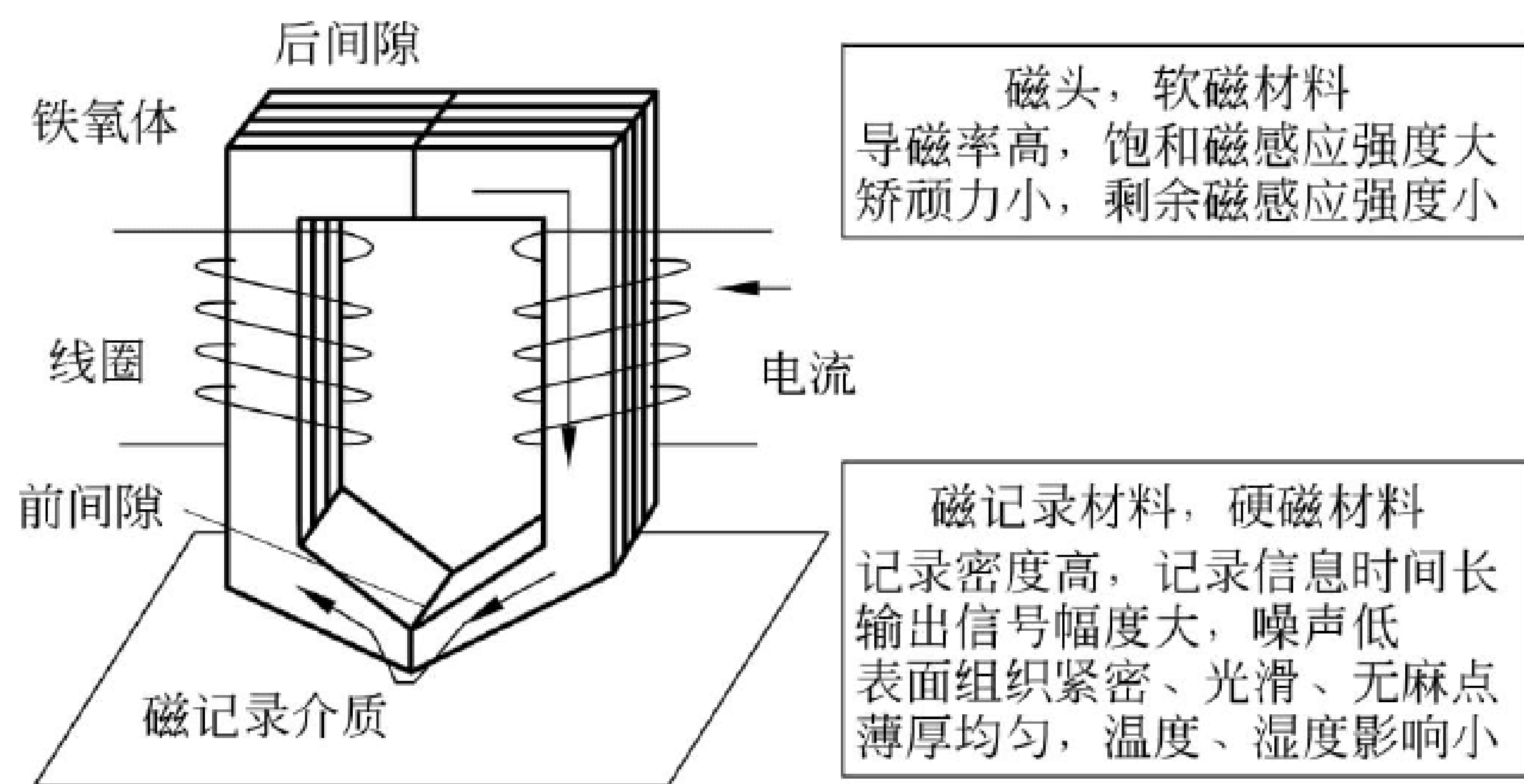


图9.1　磁头结构和电磁转换示意图

磁头是实现电磁转换过程的关键装置，通常由软磁材料(外界磁场的作用消失后，该磁性材料的磁性容易消失)做成。它是一个留有前后间隙的磁性环状物体，上面绕有线圈。后间隙越小越好，前间隙(称为工作间隙)要宽窄适当。当向线圈提供一定方向和大小的电流时，将使磁头体被磁化，建立起有一定方向和强度的磁场，即在磁环内有磁力线产生，由于磁头的前间隙处磁阻较大，将产生漏磁，这漏磁就是向磁记录介质中写入信息的信息源。

磁记录介质是在某种刚性(如硬盘)或柔性(如软盘、磁带)载体上涂有薄层磁性材料的物体，用于记录以磁状态表示的信息。磁记录介质用硬磁材料(外界磁场的作用消失后，该磁性材料的磁性尽量多地被保留)做成。

磁存储器的写入过程：当磁头前端与磁记录介质距离很近时，磁头前间隙处的漏磁将把处于附近的磁记录介质上的一小片磁性材料磁化，而当磁头离去时，就在这一小片被磁化的磁性材料区域保留了磁化状态，从而记录下写入的一位信息。若磁头线圈中无电流，磁头就不会被磁化，也就不会产生漏磁，则不会对磁记录介质产生任何影响，即无写入操作。

磁存储器的读出过程：当磁头前端与磁记录介质距离很近且高速经过时，若所经过的磁记录介质上的一小片磁性材料已被磁化，这一磁化状态将在磁头的环体内产生磁力线，从而在磁头的线圈中感应出一个脉冲电流，这表示读出了记录在磁记录介质中的一位信息。若磁头所经过的磁记录介质上的磁性材料未被磁化，则磁头也就感应不出任何磁状态的变化(实际上，正常读写过程中是不会遇到这种情况的)。应该说，磁头能感应到的是磁记录介质中磁化状态的变化情况。

2. 磁记录方式

磁记录方式指的是一种编码方法，即按什么方案(规律)，把一连串的二进制信息变换成存储介质磁层中的一个序列的磁化翻转状态，并且可以容易、可靠地用读写控制电路实现这一转换过程。选用不同的记录方式，对磁表面设备的性能有重要的影响。评定一种记录方式优劣的标准主要包括以下3个方面。

(1) **编码效率**。指记录密度与最大磁化翻转密度之比，即为记录一位信息所用的最多

磁化翻转次数的倒数。若记录一位信息,最多要有一次磁化翻转,则编码效率最高,为100%。编码效率直接影响可用的记录密度和设备的最大存储容量。

(2) **自同步能力**。指从读出的数据信息中提取出同步时钟信号的难易程度。同步时钟信号是分隔出连续多个数据的不同位所必需的时间基准信号,若能从读出的数据信息中提取出这一同步时钟信号,则称折中编码方案有自同步能力,这对某些高记录密度的系统来说是希望具有的。否则,只能用另外的办法(如单独设立提供同步时钟信号的磁道)来得到同步时钟信号,这被称为外同步。

(3) **可靠性问题**。前面已经提到,外存储设备的可靠性是一个要重点关注的问题。好的编码方案应对提高外存储设备的可靠性有所体现。在外存储设备中,往往采用能检查错误,甚至自动纠正某些最常遇到的错误的必要措施。

当前最常用的基本编码方式有如下几种,各自的写入电流和磁化强度的波形如图9.2所示。

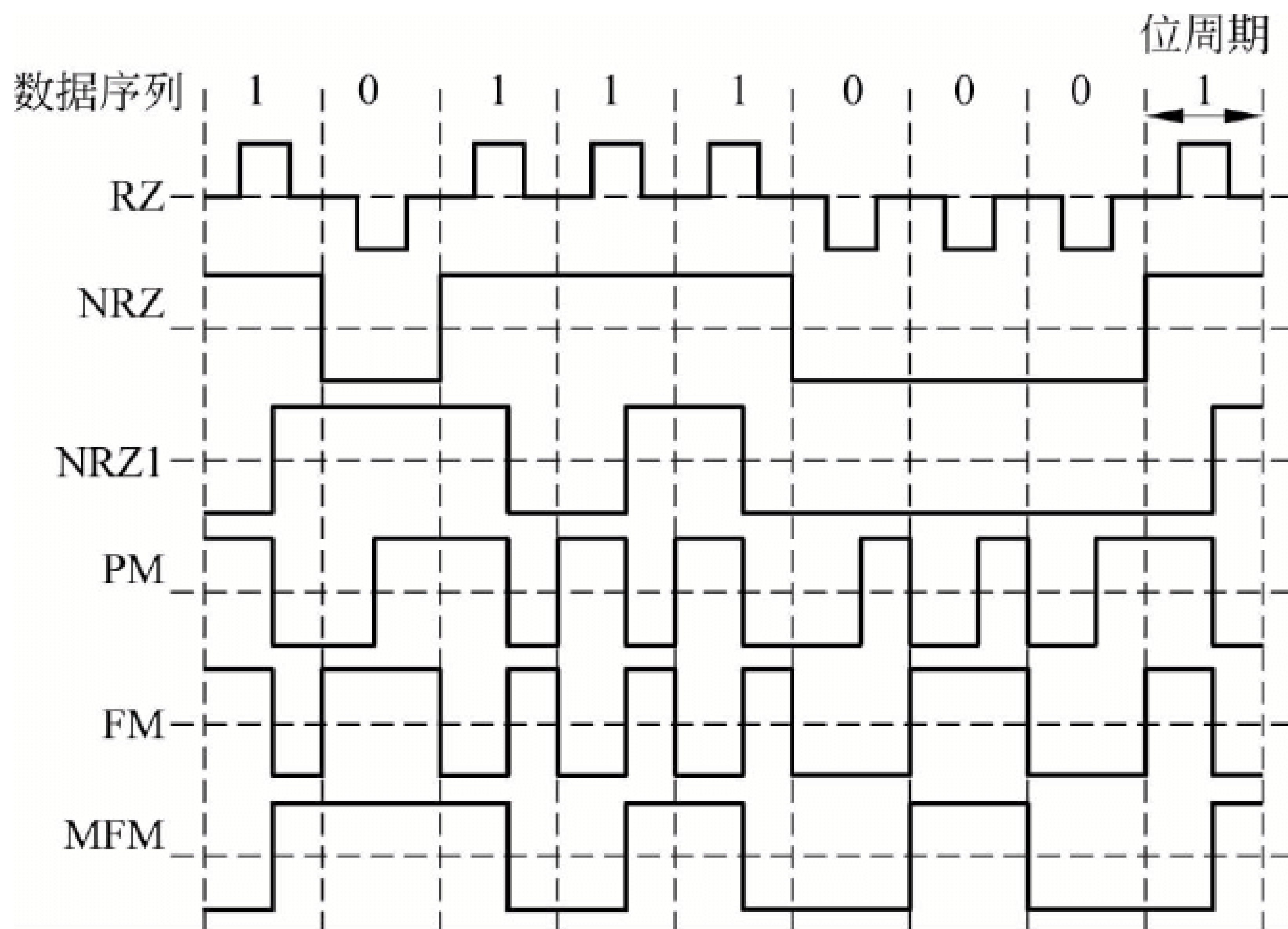


图9.2 不同记录方式的写入脉冲和磁化强度波形图

1) 归零制(RZ)

这是用向磁头线圈送入正、负脉冲电流的办法执行写“1”、写“0”操作的方案,使1和0信号的磁化状态正好相反。它的主要矛盾是在两个信息位之间磁层处于非磁化状态,难以解决,故不实用,但对理解经过电磁转换在磁性材料中记录二进制信息的原理是有帮助的。

2) 不归零制(NRZ)

这是用向磁头线圈送入正、反向电流的办法执行写“1”、写“0”操作的方案,使1和0信号的磁化状态(极性)正好相反。与前一种方案相比,取消了两个信息位之间磁头线圈中无电流的情况,故磁层中不存在未被磁化的状态,不是被正向磁化,就是被反向磁化。

3) 见1就翻的不归零制(NRZ1)

这是用在写“1”时就变化磁头线圈中的电流方向(写“0”则不变电流方向)的办法执行写“1”、写“0”操作的方案。

4) 调相制(PM)

这是用在磁层中不同的磁化翻转方向来区别数据“1”和“0”的方案,为此,磁头线圈中的电流,在写“1”和写“0”时要朝不同的方向变化,读出时,就表现为读出的信号是正还是负脉

冲，即二者的信号相位差为180°。

5）调频制（FM）

这是用在磁层中不同的磁化翻转次数来区别数据“1”和“0”的方案，记录“1”比记录“0”磁化翻转频率要多一倍。为此，磁头线圈中的电流，在每个位周期起始处要变化一次方向，在写“1”时，还要在位周期中心处再变化一次方向，而写“0”则不会在位周期中心处变化电流方向。读出时，读出的1信号表现为两个脉冲，读出的0信号表现为一个脉冲。二者的读出脉冲频率相差一倍。

6）改进调频制（MFM）

正像它的名字所指明的，这是对前面讲的调频制一种改进方案，其目的是提高这一方案的编码效率，使其从调频制的50%提高到现在的100%。这一改进表现在取消了大部分的在每个位周期起始处的改变磁头线圈中的电流方向的动作，只保留在连续的“0”信号的每个位周期起始处的电流方向变化，以便保证该编码方式的自同步能力。

9.2 磁盘设备

磁盘设备是计算机系统中最主要的外存设备，也是计算机外围设备中，在提高性能和降低成本两个方面取得骄人成绩的最典型的设备。目前使用最多最广的是硬磁盘，硬磁盘存储容量大，读写速度快，普及型产品价格也比较便宜。典型的硬盘结构是温彻斯特磁盘，简称温盘，它的主要特点是密封组合式，即磁头组和盘片组件被封装在净化腔体内，通常盘片的每面都有一个读写头。磁盘设备通常包括磁记录介质、磁盘驱动器、磁盘控制器3个组成部分。硬磁盘的逻辑结构如图9.3所示。

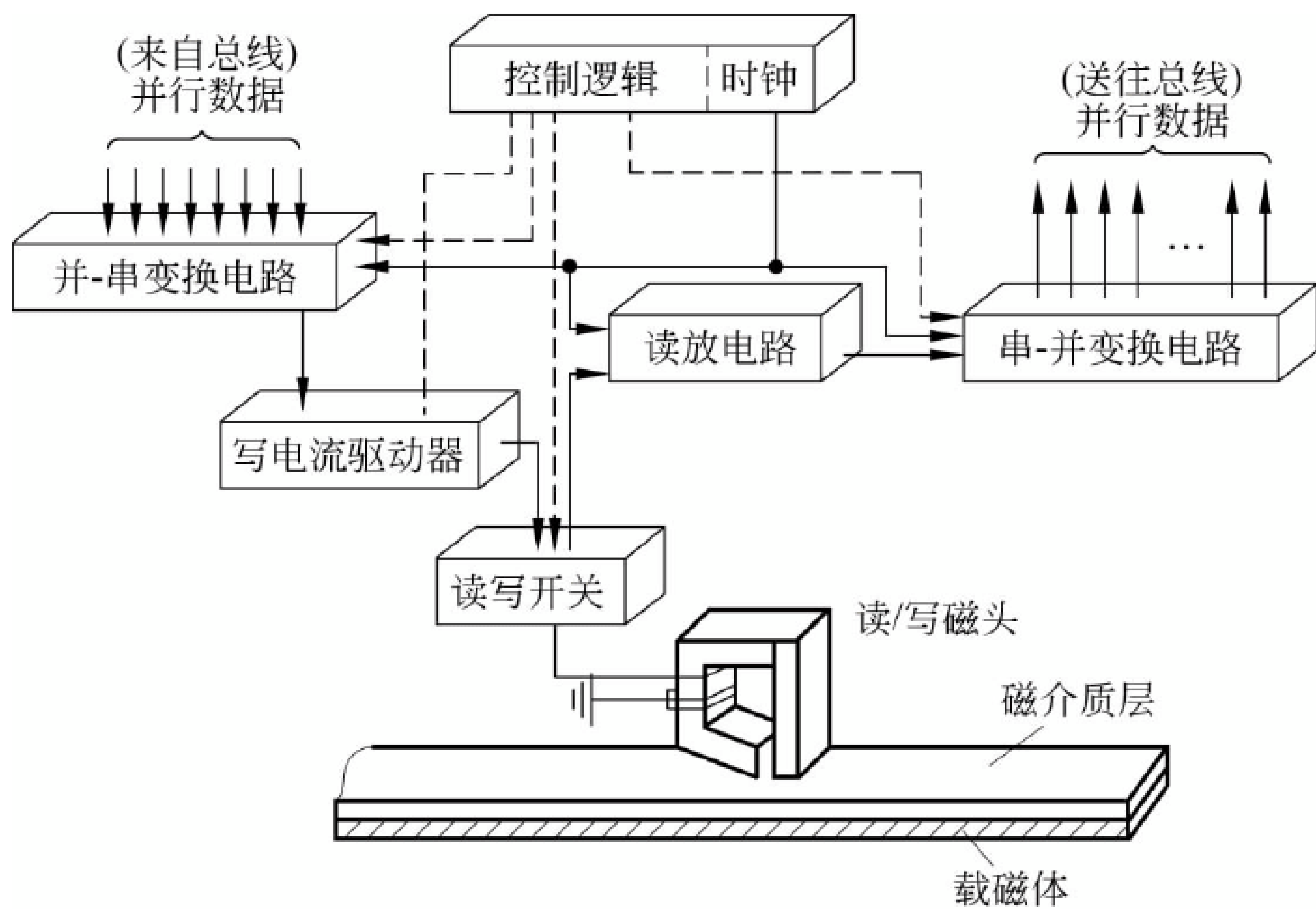


图9.3　硬盘存储器逻辑结构图

9.2.1　磁记录介质

硬磁盘盘片由一个或多个表面涂有磁性材料的铝质平盘组成。早期的铝盘直径达50cm，但现在一般也就是3～12cm，用于笔记本计算机的磁盘直径已经在3cm以下，而且还

在缩小。一般盘片的上下两面都能记录数据,通常把磁盘片表面称为记录面。记录面上存储数据位的小磁元构成的一系列同心圆称为**磁道**。多个盘片的同一半径位置上的磁道构成一个柱面。每个磁道可以划分为固定长度的**扇区**,如图 9.4 所示是磁道几何示意图。

每个扇区一般有可存放 512 字节的数据区,数据区前有用于在读写之前对磁头进行同步的**前导区**,数据区之后是纠错码(Error-Correcting Code,ECC)。连续的两个扇区之间是**隔离带**。一些制造商以未格式化时的容量来标识他们的磁盘,但多数的制造商给出的是格式化后的容量,没有把前导区、纠错码和隔离带的容量作为数据区容量计算在内。格式化后的容量一般比未格式化时的容量少 15%左右。

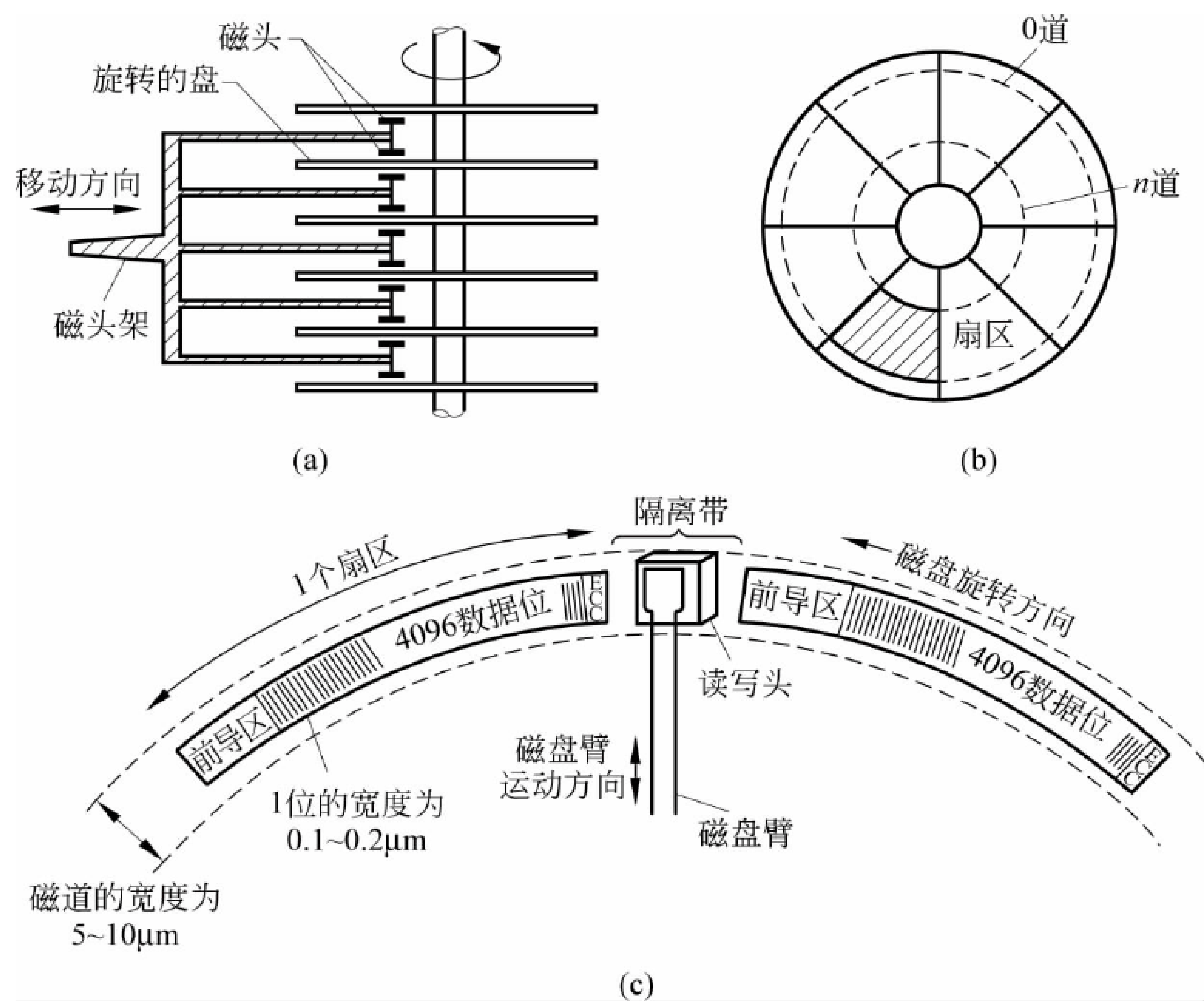


图 9.4　磁盘内部示意图

9.2.2　磁盘驱动器

磁盘驱动器是一种精密的电子和机械装置,包括作为磁记录介质使用的磁盘片和驱动磁盘匀速旋转的动力与驱动部件,完成读写功能的磁头和驱动磁头沿磁盘径向方向运动和准确定位的部件,以及其他一些控制逻辑电路等部件。图 9.5 是磁盘驱动器结构示意图。

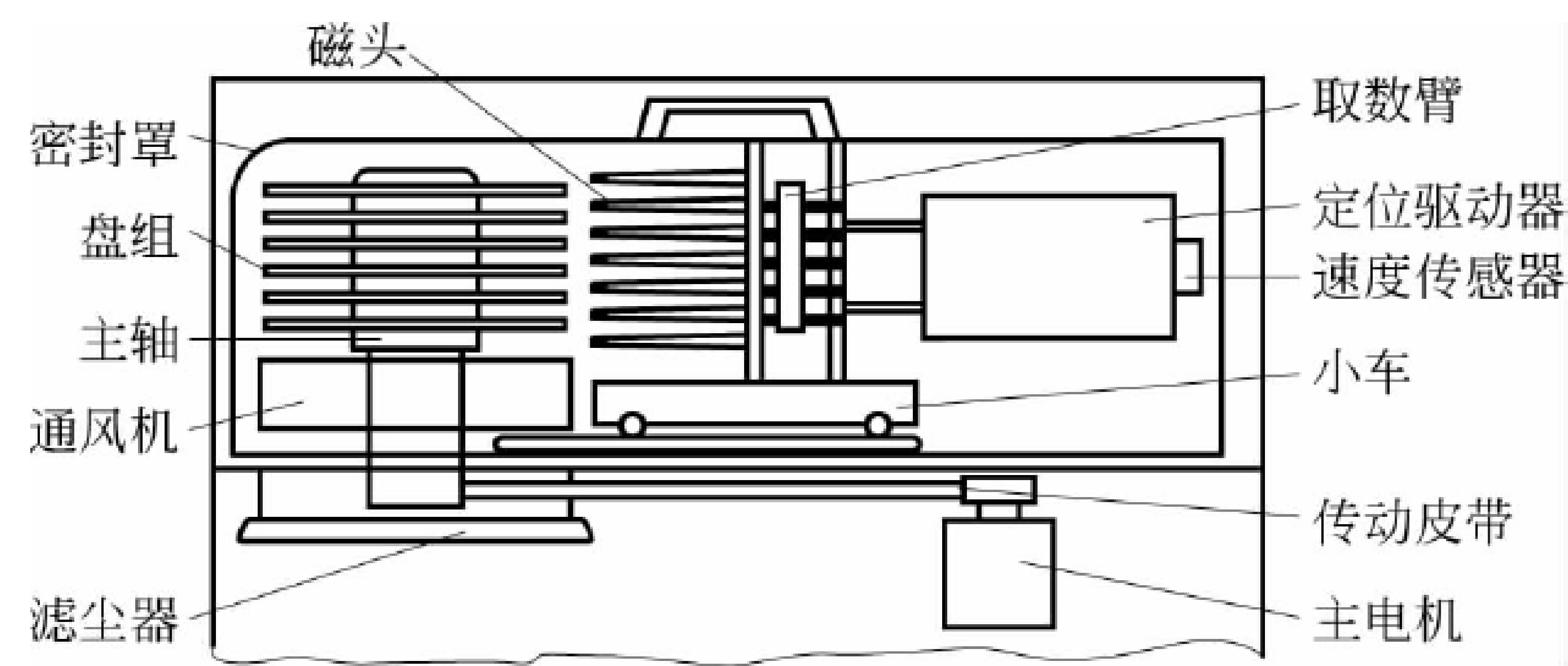


图 9.5　磁盘驱动器结构示意图

1. 主轴及其驱动系统

硬磁盘的盘片被固定在磁盘驱动器的主轴上，由主轴带动磁盘匀速旋转。磁盘驱动器的主轴是由一个主电机通过传动皮带带动旋转的。主电机的转速必须被监测和进行自动调节。为保证有正常的读写信号，保证浮动磁盘头与磁盘表面有合理的距离，要求磁盘以一个额定的转速匀速旋转。因此，必须在刚加电启动时，检测磁盘是否已达到额定转速，通常在未达到这一转速之前，可以不允许磁头进入磁盘外沿之内；在磁盘正常旋转的过程中，还可以用一个闭环的自调节系统使其转速尽可能地均匀。

2. 磁头及其定位系统

大部分的磁盘系统中，只为每个磁盘面设立一个磁头，为在磁盘径向方向的某个位置，即某一磁道，完成读写，必须驱动磁头移到并定位在那里，实现这一功能的部件被称为磁头定位驱动机构。它由磁头小车和驱动部件组成，磁头被安装在小车上，小车的运动带动磁头沿磁盘的径向方向前进或后退。这样的磁盘系统被称为活动头磁盘，它的寻找磁道的时间比较长。也有一些磁盘，出于提高读写速度的需要，为每个磁道分别安装一个或者多个磁头，使用多个磁头能够消除磁头寻找磁道的时间，这样的磁盘系统被称为固定头磁盘。

3. 数据读写等控制逻辑部分

要读写磁盘上的信息，首先必须给出信息在磁盘设备上的准确位置，这个位置信息通常由具体的磁盘面，具体的磁道的哪一个存储区域等几部分组成。

对于写操作，首先要把写入信息的地址送入磁盘的地址寄存器，然后磁盘的读写控制逻辑电路对要写入的信息进行编码处理，经过写入驱动器再送入选定的磁头的写入线圈，把信息串行地写入选定的磁道中；写入操作伴有比较完善的出错检查，并同时把这些检查结果的信息(如 CRC 码)也写在特定的存储区，用于读操作时复核读出结果的正确性。

对于读操作，首先使磁头移动到由磁盘地址寄存器指定的存储区域，选中的磁头执行读操作，读出信号送读出放大器，经译码电路分离出数据脉冲，拼装成字节或字的格式送入磁盘接口。读的过程也伴有出错检查，甚至是自动纠错等操作。

9.2.3 磁盘控制器

磁盘控制器即磁盘驱动器适配器，是计算机主机与磁盘驱动器之间的接口设备。它接收并解释计算机主机发来的命令，向磁盘驱动器发出各种控制信号。检测磁盘驱动器状态，按照规定的磁盘数据格式，把数据写入磁盘和从磁盘读出数据。

磁盘控制器类型很多，但它的基本组成和工作原理大体上是相同的。磁盘控制器主要由与计算机系统总线相连的控制逻辑电路，微处理器，完成读出数据分离和写入数据补偿的读写数据解码和编码电路，数据检错和纠错电路，根据计算机主机发来的命令对数据传递、串并转换以及格式化等进行控制的逻辑电路，存放磁盘基本输入输出程序的只读存储器和用以数据交换的缓冲区等部分组成。

现在微型计算机的磁盘系统中应用最为广泛的磁盘接口有 IDE、SCSI 和 SATA 等。

IDE 和 EIDE 接口在微机中得到了广泛的使用。普通 IDE 的特点是，数据传输率小于 1.5MB/s，最多连接两个 IDE 设备(硬盘或其他设备)和每个 IDE 硬盘容量小于 528MB。而 EIDE 的特点是，数据传送率为 2～18MB/s，最多可连 4 个 IDE 设备，EIDE 硬盘容量可以超过 528MB，同时支持逻辑块寻址(LBA)，并能控制 CD-ROM 驱动器。EIDE 控制器通

常接在 VESA 总线和 PCI 总线上。

SCSI 是用于小型机和微型机的外部设备接口。不同版本的 SCSI 的数据宽度为 8～32 位,数据传输率为 5～40MB/s,SCSI 接口最多可连接 1～7 个 SCSI 设备。除了硬盘外,还可以是 CD-ROM、CD 刻录机、扫描仪、磁带机或其他 SCSI 计算机外设。每个设备有两个插口,一个用于输入,另一个用于输出。每个 SCSI 设备有一个唯一的 ID,编号从 0～7。常见的 8 位数据 SCSI 电缆线有 50 线,其中地线(有利于噪声屏蔽)25,数据线、校验位、控制、电源和预留 25,16/32 位数据设备需第 2 根电缆提供其他信号。

9.3 磁盘阵列

作为计算机系统外存储器的主要支柱设备,磁盘的容量、读写速度、价格和容错支持,一直是人们致力解决的问题。过去很长一段时间,研究工作多集中在提高、改善单个个体磁盘机的性能方面,因为待改善的性能、要做的技术工作都有较大的选择余地,这些工作也确实取得了巨大成绩,极大地推动了磁盘的普及应用。但这是不是解决问题的唯一途径呢? 显然不是,另外一个可行途径,是使用统一管理的由多个磁盘组成的磁盘阵列。

磁盘阵列技术最早由美国的一个科研小组提出来,并且很快成为被工业界广泛接受的一项技术。这一技术的着眼点,还是通过多个磁盘设备的并行操作来提高设备总体的性能和可靠性。显而易见,如果一个磁盘有 x MB 的容量,单位时间提供 y MB 的传送能力,则概念上讲,n 个这样的磁盘就有($n \times x$)MB 的容量,($n \times y$)MB 的传送能力,换句话说,要读出 y MB 的数据,所用的平均时间只要原来单个磁盘所用时间的 $1/n$;此外,还有一点好处是,通过合理地在多个磁盘之间组织数据,可以得到比较理想的容错能力,这指的是,额外拿出一定的存储容量(冗余),用于保存检错纠错的信息。

从总体价格上考虑,使用多个磁盘也并不会给用户带来太大的经济负担。因此,该科研小组在提出这一技术思路时,用的词为 Redundancy Arrays of Inexpensive Disks(RAID),但到了工业界,却更愿意把这里的 Inexpensive 换成 Industry,在原来技术概念的基础上加上点商业味道。

为统一管理磁盘阵列,使用户所感觉到的不再是多个物理盘,似乎就是一个性能更高的单个磁盘,就要使用一块特定的接口卡(一般称为 RAID 卡,阵列控制卡),把组成阵列的多个物理磁盘连接为一个逻辑整体,这被称为一个逻辑磁盘,如图 9.6 所示。该卡的一端将被插接到高速的 SCSI 总线或 PCI 总线的插槽中,以便与计算机主机接通,另外一端有 1～3 个接插头,通过电缆与 1～3 组磁盘设备连接,每组可有串行连接在一起的 1～7 物理磁盘。该卡是一个有较强处理能力的接口电路,上面有一个单片计算机,形成奇偶校验信息的机构,分析与处理主机 CPU 发送来的读写磁盘命令的机构,有起缓冲作用的 DRAM 存储器(又被称为阵列加速器,几 MB 到几十 MB 容量,分成两个体以镜像方式运行,还有专用的后备电池支持)等几个组成部分。系统能通过该卡对连接到卡上的多个磁盘,按用户的使用要求,灵活地配置为不同的使用和容错方式。

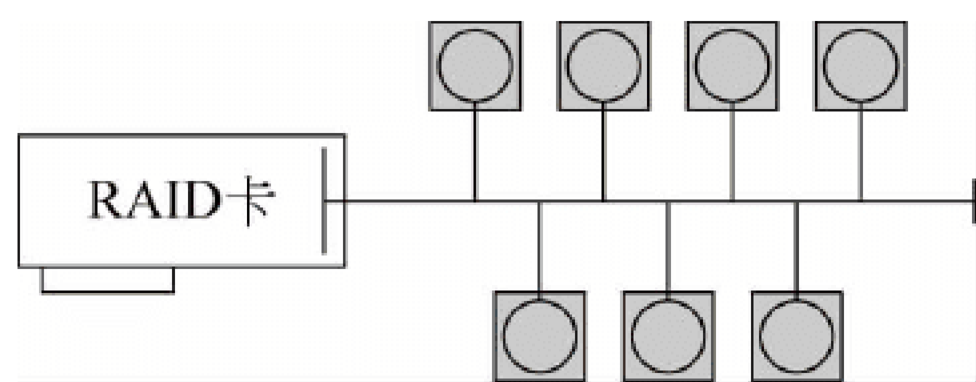


图 9.6　磁盘阵列示意图

阵列磁盘运行过程中,有两项重要技术对磁盘系统的运行性能产生较大影响。一个是

并发命令请求和命令排队，就是说CPU可以向磁盘设备发送多条命令，阵列卡会对这些命令进行排队管理，并使多个命令得以并发处理；如果在处理命令的时候，还能进行某些性能优化，而不是机械地按命令到来的先后次序处理，还可以进一步提高数据读写的速度。最简单的例子就是对两个等待读操作的命令，磁头先到达哪一个命令的数据扇区，就先执行哪一个命令，这在磁盘本身的控制器部分来处理可能更方便。另外一项技术是设备的快速接入和断开，即当一个占据了总线的磁盘开始执行一个读命令，数据又尚未准备好时，它应快速地暂时把自己从总线上断离出来，以便使另外正急于使用总线的磁盘可能抢到总线，从而提高总线的使用效率和系统性能；当这个磁盘准备好数据时，应保证它能把自己尽快地接通到总线上去（得到总线的使用权）。这实质上是把占用总线的时间压缩到尽可能短的一项处理技术。

阵列磁盘的一个重要的特性就支持容错。合理地把一个文件的内容划分为“块”并写到组成一个逻辑磁盘的多个物理磁盘中去，再采取适当的数据存储保护措施，不仅可以提高数据读写的速度，而且可以大大增加磁盘系统工作的可靠性，就是说使该磁盘系统具有很好的容错能力。提出这一技术思路的研究人员把这一容错划分成6种模式，又经常被说成6级容错，分别叫作RAID0、RAID2、…、RAID5。其中的RAID2方案与磁盘设备本身的工作特性不完全符合，RAID3要求多个物理磁盘同速并保持相关扇区同步，难以得到好的性能/价格比，较少采用，其他4种已被工业界广泛接受并在一些产品中得到实际应用。

RAID0模式是指把一个文件的数据分成容量相等（例如16KB）的“块”（Chunk），把每一段交替地分别写到不同的物理磁盘的几个扇区中去，这种处理叫Data Striping。它的好处是，不仅可以使几个磁盘合起来有更大的容量，还可以让多个物理磁盘并发读写，提高了数据输入输出的吞吐率；它并没有采用任何容错措施，故没有容错能力。磁盘可用存储容量全部用于存储实际数据，如图9.7所示。

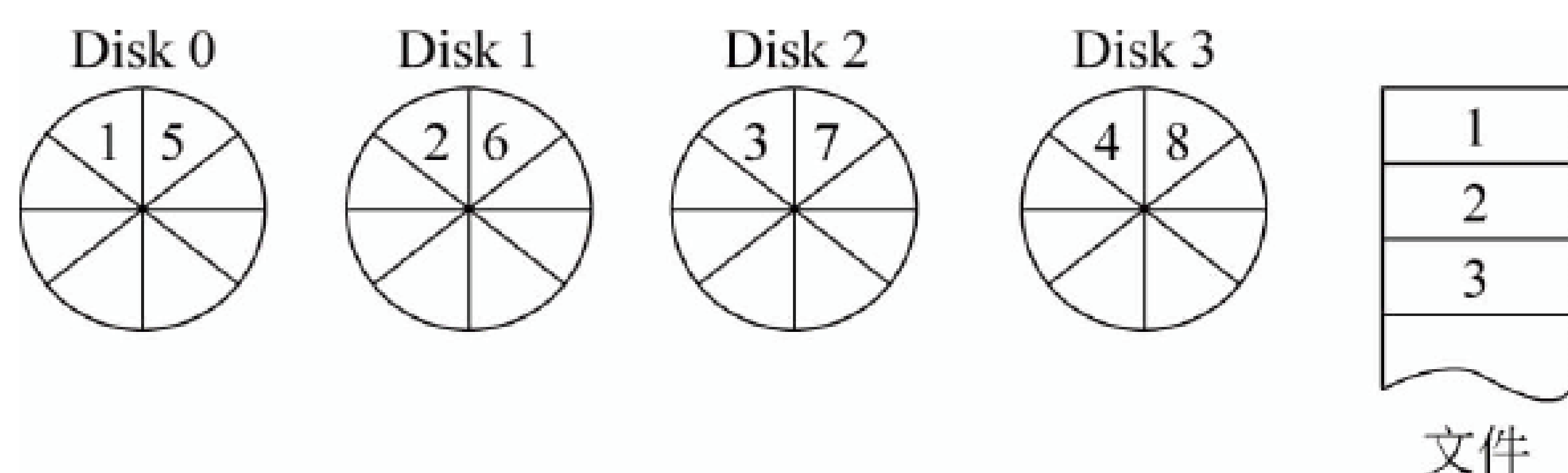


图9.7 RAID0，Data Striping

RAID1模式是实现两个磁盘互为备份的用法，即把相同的数据分别写到这样配对使用的两个磁盘中去，这叫作磁盘镜像（Drive Mirroring），这一写操作是对两个磁盘同时进行的，不会有降低写入速度的矛盾。它的最大好处是数据被同时保存在两个磁盘中，若其中任何一个磁盘出现故障，可以从另一个磁盘中读出数据，而不会出现令人难以接受的丢失数据的局面。读操作时，也许还会碰上读出数据略快一点的情况。它的不足之处是，镜像磁盘总存储容量的有效利用率只有50%。



图9.8 RAID1，Drive Mirroring

这里的镜像是完全用硬件实现的，运行速度快；磁盘镜像也可以用软件（操作系统）实现，甚至于用在仅有一个磁盘的计算机系统，其运行速度要大受影响。在有硬件镜像的系统

中,不要再指定软件镜像,否则会把有关镜像的数据文件都存成4份,系统运行速度和所使用的磁盘空间都难以令人接受。

RAID4模式是为 N 个存储数据的磁盘分配另外一个专用于存储奇偶校验信息的磁盘,它仍以Data Striping为基础,但在把文件数据分块写进多个数据磁盘的同时,对这些数据中相应的几位求出它们的奇偶校验值,最终形成一个由奇偶校验值组成的信息快,并将其写入专用于存储奇偶校验信息的磁盘,这被称为数据保护(Data Guarding)。它的好处是提供了容错能力,即这 $N+1$ 个磁盘中任何一个出现故障,都不会造成丢失数据的问题,可以用剩下的 N 个磁盘的内容,计算出存放在有故障磁盘中的正确的数据内容,尽管这一计算比较费时间。与磁盘镜像相比,它的存储容量的有效利用率可以更高,为 $N/(N+1)$。这种模式下,最少的物理磁盘数量为3。它的缺点,一是用运行正常的 N 个磁盘的内容计算出有故障磁盘中的正确数据比较费时间;二是受奇偶磁盘的制约,不支持多个数据磁盘的并行写操作,如图9.9所示。

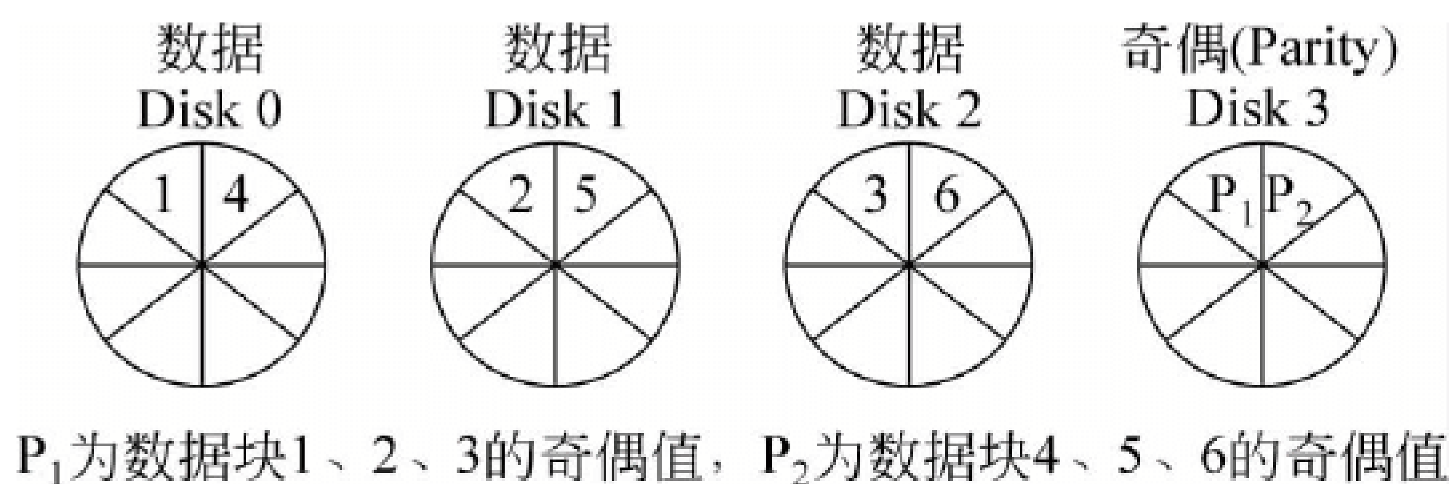


图9.9 RAID4,Data Guarding

RAID5模式是对RAID4的改进。这一改进表现在:不再区分 N 个存储数据的磁盘和另外一个专用的奇偶校验磁盘,它是把 $N+1$ 个磁盘同等对待,都用于存放数据和奇偶校验信息,在同一个物理盘中,数据和奇偶校验信息是以不同扇区的形式体现出来的,这被称为分布式数据保护(Distributed Data Guarding),如图9.10所示。与RAID4同样,它提供了容错能力,即这 $N+1$ 个磁盘中任何一个出现故障,都不会造成丢失数据的问题,可以用剩下的 N 个磁盘的内容,计算出存放在有故障磁盘中的正确的数据内容,存储容量的有效利用率同样为 $N/(N+1)$。这种模式下,最少的物理磁盘数量为3。在一些情况下,可能可以对多个磁盘执行并行写操作,因为它不再受单独一个奇偶磁盘的制约。

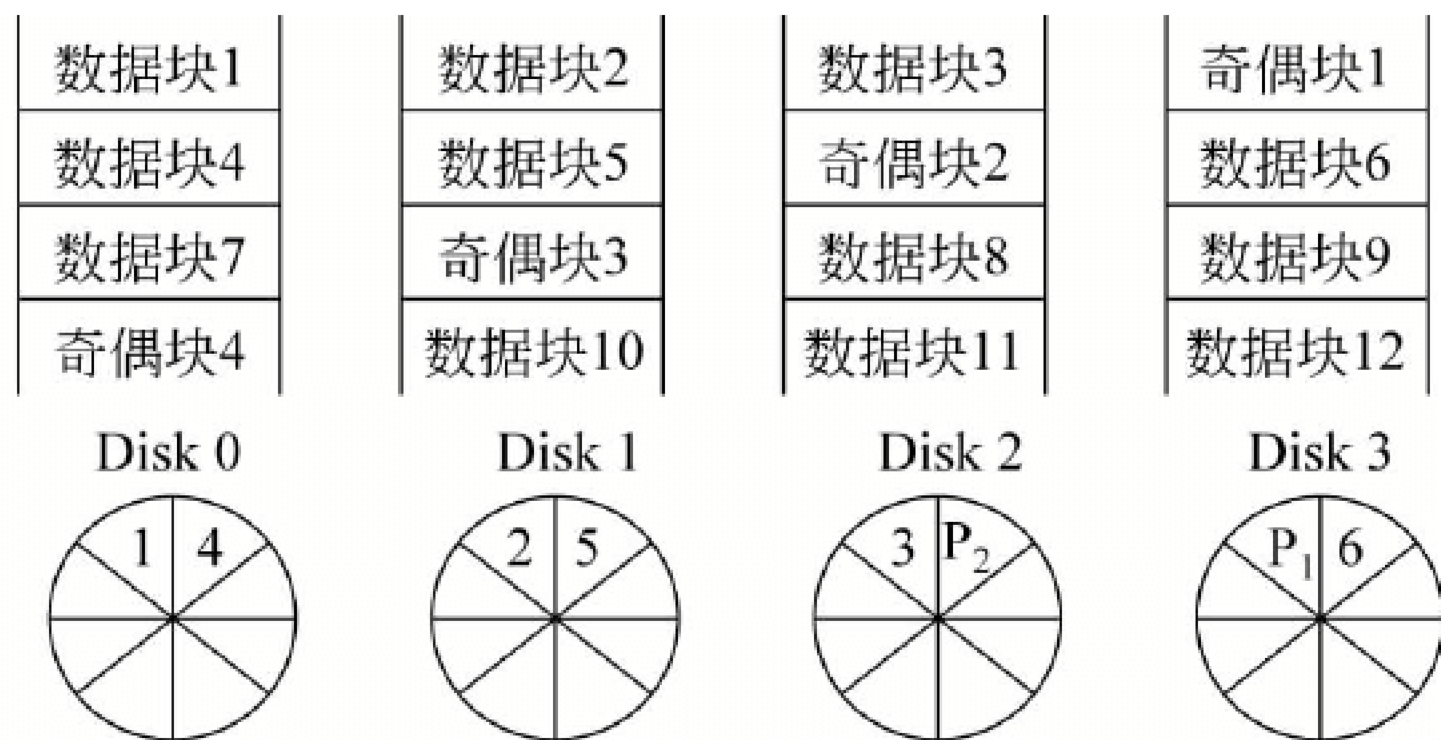


图9.10 RAID5,Distributed Data Guarding

除此之外,阵列盘技术还支持联机热备份磁盘,即把一台磁盘接入计算机系统中,平时它并不执行读写操作,只以备份者的身份出现,当带有容错支持的阵列盘中某一磁盘出现故障时,就令这台备份磁盘立刻“上岗”。首先把出现故障磁盘中的数据复制(会用到容错支

持)到已“上岗”的磁盘中,之后用这台磁盘顶替已出现故障的磁盘继续投入运行。接下来可以把有故障的磁盘从系统中取下来进行维修。有容错支持和配有备份磁盘的系统中,从发现磁盘故障到启用备份磁盘、恢复故障磁盘中的数据到备份磁盘、停止有故障磁盘的运行而用备份磁盘顶替它的这一完整过程,若都经由计算机系统的硬软件自动执行(无须操作或管理人员干预),这被称为数据的自动恢复,通常要用几到十几分钟的时间完成。

若在系统不断电,磁盘设备也不断电的情况下,可以直接把一台设备从系统中拔下来或接通上去,这叫热插拔技术。它必然只能用在带有容错支持的阵列盘系统中,需要有相应软件、总线、接口和设备几个方面的支持。

9.4 光盘设备

20 世纪 80 年代,出现了一种新的存储介质——光盘。光盘存储密度高于普通的磁盘,原本是为存储电视节目而开发的,但作为计算机的存储设备,它能发挥更好的作用。由于其潜在的巨大存储能力,对光盘的研究已经是一个重要的课题,从最早的只读光盘发展到后来的可刻光盘、可擦写光盘,已经取得了巨大的进步,光盘的容量以及光盘驱动器的读写速度都有了大幅度的提高。下面对几种不同类型的光盘分别做一下简单的介绍。

9.4.1 只读光盘

第一代光盘是由荷兰的电子企业集团飞利浦公司发明的,用来存放电影。大小为 30 厘米左右,以激光影碟的名称面市。1980 年,飞利浦和索尼一起推出了光盘(Compact Disc, CD),并很快取代了乙烯基唱片。CD 的详尽技术细节以正式的国际标准(ISO/IEC 10149)公布,并因为封面的颜色被通称为**红皮书**。国际标准规定所有的 CD 大小为 120mm 左右,厚度为 1.2mm 左右,中心有一直径为 15mm 的孔。

CD 是通过在涂有玻璃表层的主盘上,用高能红外激光束烧出 0.8mm 直径的小孔制成的。用这种主盘做成模子,上面带有烧好的激光孔,然后往模子上注入熔化的多种碳酸盐脂,使激光孔的形状和玻璃主盘的形状一样,就基本上完成了 CD 的主体。接着,在碳酸盐脂上沉淀上一薄层的反射铝,再覆盖上一层起保护作用的表层,最后再打上标签,整个 CD 就完成了。碳酸盐脂底基的凹陷部分叫作**凹区**,凹区两边未经过烧制的部分叫作**凸区**。

将 CD 进行回放时,用一个低能激光二极管发出的波长为 0.78mm 的红外光照射在二极管下“流过”的凹区和凸区。光源在碳酸盐脂层的上方,所以,当凹区经过时,激光束就会比凸区经过时伸出一些。由于凹区的高度为激光波长的 1/4,从凹区反射的激光的波长为从凸区反射光的波长的一半。这样,反射光和发射光叠加,将导致光接收器接收到的从凹区反射的光线比从凸区反射的要弱。CD 机通过这种途径,可以区别出凹区和凸区。虽然用凹区代表 0,凸区代表 1 可能是最简单的表示方法,但从可靠性方面考虑,用凸区/凹区和凹区/凸区转换来表示 1,而用连续的凹区或凸区来表示 0 的可靠性要高一些,所以,CD 上采用的是这种模式。

凹区和凸区写在一根单向的螺旋线上,螺旋线从靠近孔的地方发出,一直到离盘边 32mm 处,如图 9.11 所示。螺旋线在盘上共有 22188 圈(每毫米约 600 圈),如果没有损坏的话,长度为 5.6km。

为使光盘上的音乐以恒定的速度播放,就必须保证凹区和凸区"流动"的线速度保持恒定。也就是说,随着CD机的读盘头从里到外移动,CD盘的旋转速度必须持续下降。在里圈,旋转速度应为530r/min,以达到理想的线速度——120cm/min;到外圈后,要使读盘头保持同样的线速度,旋转速度应降到200r/min。保持恒定线速度的光盘驱动器和硬盘驱动器有很大区别,硬盘驱动器不管磁头在什么位置都保持恒定的角速度。而且,530r/min的角速度和大多数硬盘驱动器3600~7200r/min的角速度也有很大的差别。

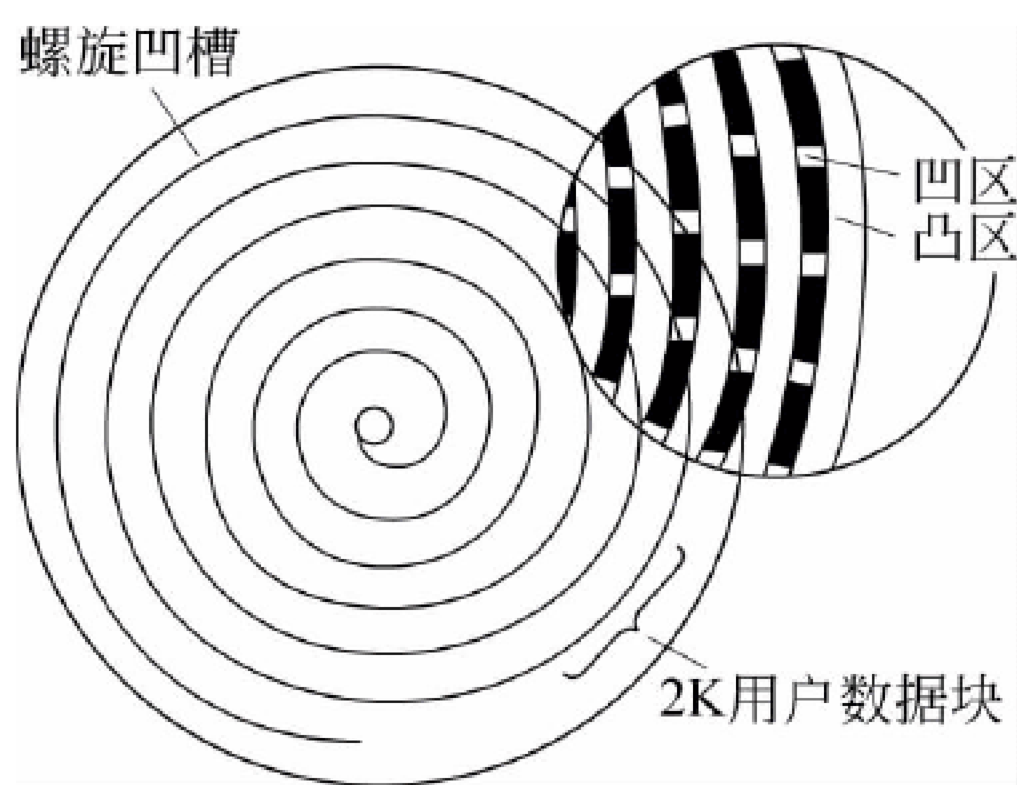


图9.11 光盘和只读光盘的记录结构

1984年,飞利浦和索尼认识到用CD存放计算机数据的潜在可能,共同出版了**黄皮书**,精确定义了现在被称为**只读光盘存储器**(Compact Disc-Read Only Memory,CD-ROM)的CD的标准。为了占领当时已经很大的音频CD市场,CD-ROM采用了和音频CD一样的外形、大小,在机械和光学两方面和音频CD兼容,甚至采用相同的铸模机生产。

黄皮书中定义了CD-ROM中计算机数据的格式,它提高了系统的纠错能力,这对CD-ROM来说十分重要。因为音乐迷们可能并不太在意音频CD中这儿或那儿丢了一位,但计算机用户对此却十分挑剔。CD-ROM的基本数据格式将每个字节编码成14位的符号。本书前面已经提到,14位足够将8位长的字节进行汉明编码,还有两位剩余。实际上,CD-ROM采用的是新的编码系统,将读出的14位符号映射到8位的字节是通过硬件查表进行的。

连续的42个符号一组,构成了588位的**帧**。每帧包含192位数据位(24个字节),其他的396位用于纠错和控制位。一直到此,音频CD和CD-ROM的数据存储模式完全相同。黄皮书中增加的内容是将98帧作为一个**CD-ROM扇区**,定义了两种数据存储格式。单速CD-ROM驱动器的工作速度为75个扇区每秒,对于第一种存储格式光盘的数据速率是153600B/s;第二种存储格式光盘的数据速率为175200B/s。倍速光盘的速度为其两倍,其他速度的驱动器也可以此类推。标准音频CD的容量是存放74分钟的音乐,如果采用的是第一种存储格式,容量是681984000B,也就是一般说的650MB。

1986年,飞利浦再次通过发布**绿皮书**,增加了图形标准和在一个扇区内交叉存放音频、视频和数据的能力,为多媒体CD-ROM奠定了基础。

CD-ROM在出版游戏、电影、百科全书、地图集等各种各样的著作获得广泛的应用。目前的大多数商业软件也来自CD-ROM。其大容量和低造价的完美结合将使它的应用得到进一步的推广。

9.4.2 可刻光盘

早期制作CD-ROM母盘的设备十分昂贵,但是,作为计算机行业的规律,不会有任何东西长久处于高价位。到20世纪90年代中期,CD刻盘机的大小已和普通的读盘机相差无几,并作为普通的外围设备出现在许多计算机商场中。由于一旦写过以后,CD-ROM的内容将无法擦除,这种设备和磁盘比还是有些不足。但是,刻盘机还是很快在作为大容量磁盘的备份介质、允许个人或正在起步的小公司小批量生产自己的CD-ROM或为大批量生产商

用 CD 制作母盘等许多应用中找到了生存之地。用于刻盘机进行刻录的光盘,我们称其为**可刻光盘**,也就是通常说的 **CD-R**(CD-Recordable)。

CD-R 在大小上和 CD-ROM 一样,最初时也是 120mm 的空白盘,只是 CD-R 有一条 0.6mm 宽的凹槽,用来引导激光进行刻盘。凹槽有 0.3mm 的正弦偏移,频率为 22.05kHz,用来准确控制 CD-R 的转速,并在必要时加以调整。外观上 CD-R 也和普通的 CD-ROM 相同,只是 CD-R 表面是金色,而不像 CD-ROM 那样表面是银色。表面是金色是因为 CD-R 用真正的金子代替铝来做反射层。和真正的 CD-ROM 不同,它们的表面是实实在在的凹凸不平,而 CD-R 的凹区和凸区是用不同的反射光来模拟,这点是通过在碳酸盐脂和金质反射层之间加上一层染料来实现的,如图 9.12 所示。目前使用的有两种不同的染料,一种是花青染料,其颜色是绿色的;另一种是金色染料,颜色为黄橘色。化学家们正在为这两种染料哪种更好一些吵个没完。这些染料和显像中的染料十分类似,这也是柯达和富士成为 CD-R 的主要制造商的原因。

CD-R 被刻之前,染料层是透明的,激光束可以穿过它后从金质层反射回来。刻盘时,照射 CD-R 的激光能量被调高到 8~16mW,光束照射到染料的一个点上时产生的热量使之发生化学反应,改变了染料的分子结构,产生一个黑点。读出时(激光束的能量为 0.5mW),光接收器就可以分辨出染料被照射过的黑点和未被照射过的透明区,并用这个区别来对应普通光盘的凹区和凸区,甚至在普通的 CD-ROM 或音频 CD 的读盘机上也是这样区分的。

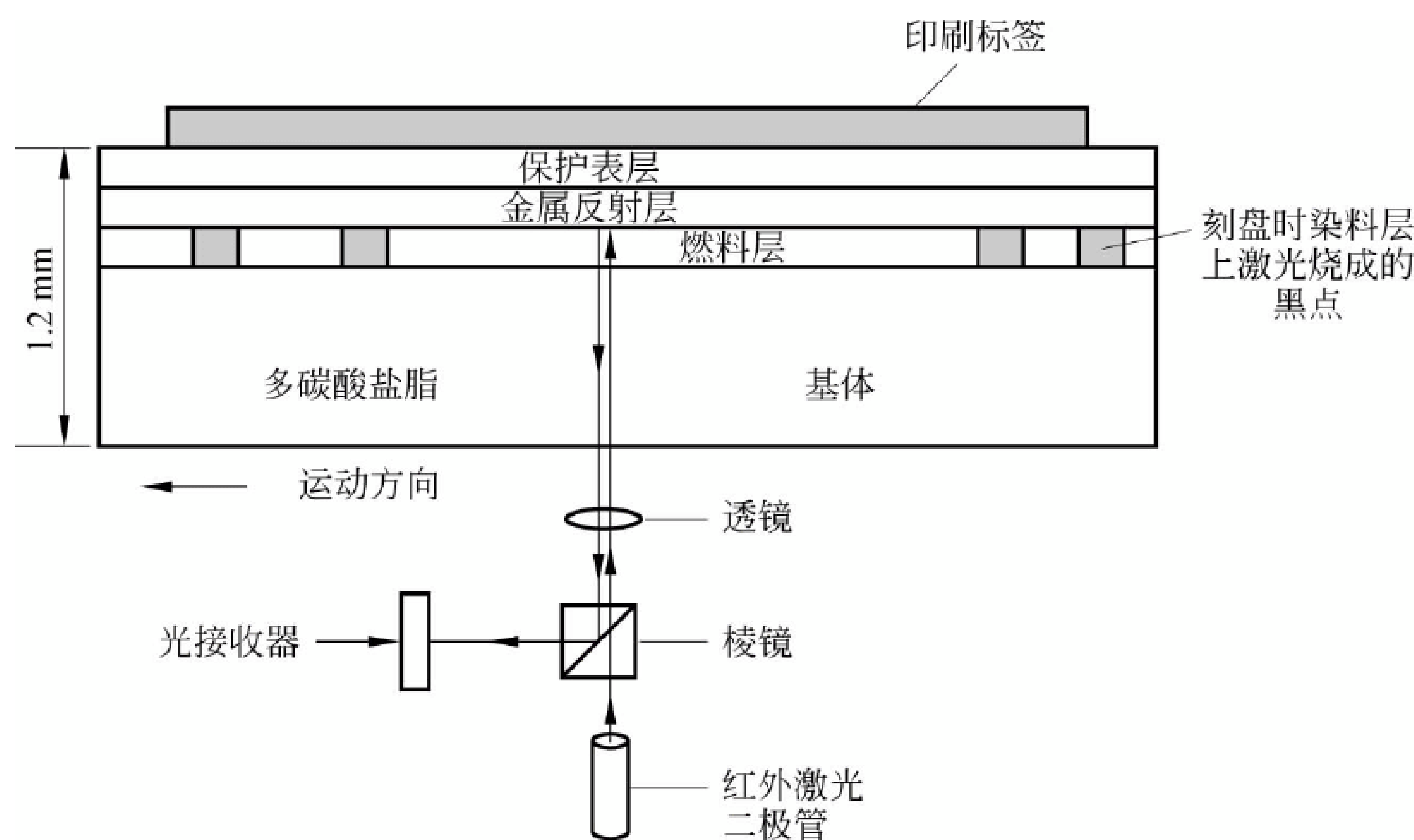


图 9.12　CD-R 光盘截面图

每种 CD 都有其引以为豪的颜色作为标准的封面,CD-R 也不例外,它的标准被称为**橘皮书**,于 1989 年出版。这个标准除定义了 CD-R 的格式之外,还定义了 **CD-ROM XA** 的格式,这种新的格式允许增量刻写 CD-R,今天几个扇区,明天几个扇区,然后下个月再刻几个扇区。一次刻在 CD-R 上的几个连续扇区被称为 **CD-ROM 道**。

CD-R 的出现使各行各业和公司能方便地对 CD-ROM(包括音频 CD)进行复制,也逐渐带来了对出版商版权的侵犯。目前,人们已经发明了一些办法来为盗版制造障碍,甚至使 CD-ROM 离开出版商提供的读盘软件就无法读出。办法之一是将 CD-ROM 上所有文件的长度写成好几个吉字节,这样,用标准的读盘软件将无法将文件拷到硬盘中,而文件的实际

长度在出版商提供的专用读盘软件中或隐藏(可能还是加密后)在CD-ROM的某个意想不到的地方。另一种办法是在选定的几个扇区中有意写入一些错误的ECC码,预计一般的CD复制软件将会自动“修复”这些错误,但光盘的应用软件却自己对这些ECC码进行检查,如果它们正确就停止执行。还可以采用非标准的道间沟或其他的物理上的“硬伤”来防止对CD的非法复制。

9.4.3 可擦写光盘

尽管CD-R使用起来很方便,价格也足够便宜,但大家还是希望有可多次写的CD-ROM。能满足人们这个需求的是**可擦写光盘CD-RW**(CD-ReWritable),其大小和CD-R相同。当然,用的染料不再是花青和金色染料,而是用银、铟、锑和碲组成的合金做记录层。这种合金有两个稳定态:晶态和非晶态,两个状态有不同的反射特性。

CD-RW的驱动器使用3种不同能量的激光。在高能激光照射下,合金熔化并从高反射性的晶态转化为低反射性的非晶态,表示凹区。在能量中等的激光束照射下,合金熔化并重新转化为本来的晶态,又成为凹区。低能激光可以感知材料的状态(用来读盘),但不会导致状态转换。

CD-RW未能取代CD-R的原因是CD-RW的空盘价格高出CD-R空盘许多。而且,对那些备份硬盘的应用来说,CD-R只允许一次写,不会被误操作清除的特性也是一大优点。

9.4.4 DVD

最初的CD/CD-ROM格式于1980年面世,从此,CD技术以日新月异的速度得到发展,到现在,大容量的光盘价格已相当低廉,市场需求巨大。好莱坞极其愿意用数字化的光盘来替代传统的影像磁带,因为光盘的影像质量高、制造成本低、可长时间保存、在商店中的架子上占用的空间比较小,而且不需要倒带。娱乐性电子公司希望能找到合适的存储介质,而许多计算机公司也想在其软件中增加多媒体特性。

这3个市场广阔而又能量巨大的行业的市场需求和技术的结合造就出了**DVD**,它最早是**数字影像盘**(Digital Video Disk)的缩写,但现在一般指**数字多用途盘**(Digital Versatile Disk)。DVD的基本设计和CD相同,也是120mm直径的注入碳酸盐的盘模,由激光二极管照射的凸区和凹区组成,通过光接收器读入信息。其新特性有:

(1) 凹区更小(DVD为0.4μm,而CD为0.8μm)。

(2) 螺旋线更紧凑(DVD道间距为0.74μm,而CD的道间距为1.6μm)。

(3) 使用红色激光(DVD激光的波长为0.65μm,而CD的为0.78μm)。

这些改进使DVD的容量比普通光盘提高了7倍,达到4.7GB。单速DVD驱动器的工作速度为1.4MB/s(而CD为150KB/s)。为了进一步提高存储容量,DVD定义了下面4种格式:

(1) 单面单层(4.7GB)。

(2) 单面双层(8.5GB)。

(3) 双面单层(9.4GB)。

(4) 双面双层(17GB)。

双层技术是在光盘底层有一层反射层,上面是一层半反射层。根据激光聚焦在哪层来

决定反射哪一层。为提高可靠性,需要将底层的凹区和凸区设计得稍微大一些,所以其容量比上层要稍微小一些。

将两张0.6mm厚的单面盘的背面互相粘在一起就制成了一张双面盘。为使所有格式的盘片厚度一致,单面盘也是由0.6mm的盘片后面粘上空白底层组成。

9.4.5 Blu-Ray

在计算机行业,没有任何技术能一直保持住其地位,存储技术也不例外。DVD刚刚推出不久,一种新的技术**Blu-Ray**就威胁要将它取代。之所以这样命名,是因为它用蓝色的激光取代了DVD用的红色激光。蓝色激光波长比红色的要短,这就使它可更精确地聚焦,能分辨出更小的凸区和凹区。单面的Blu-Ray盘可存放大约25GB的数据,双面的存储容量约50GB。数据速率约为4.5MB/s,对光盘来说这已经相当不错,但与磁盘比还相差很远。预计Blu-Ray最终将完全取代CD-ROM和DVD,但这个过程将持续数年。

本章内容小结和学习方法建议

本章主要讲解磁盘、光盘等外部存储设备。从设备本身的角度,给出了这些设备的基本组成和读写的原理性知识。从计算机整机系统的高度来看,还涉及这些设备的接口线路,以及设备与计算机总线的连接关系。从功能和使用的角度来看,讲到了磁盘阵列技术和容错、备份和热插拔、数据恢复等问题。本章内容中属于概念和一般了解的知识比较多。

习题与思考题

1. 磁盘设备的主要技术指标有哪些?
2. 磁记录方式指的是什么?有哪几种常用的编码方式?各自的编码效率与自同步能力如何?
3. 简单说明硬磁盘驱动器的组成。
4. 为什么格式化后的可用容量会比可存储的总信息位少了许多?
5. 简单比较温式硬盘设备和常用光盘设备在使用场合和性能方面的同异之处。
6. 简单说明可刻光盘的组成和实现数据读写的原理性过程。
7. 只读光盘、可刻光盘、可擦写光盘的运行原理有什么不同?盘表面的材料特性有什么区别?
8. 为什么要采用磁盘阵列技术?何为逻辑盘,何为物理盘?作为一个逻辑盘使用的多个物理盘需要在转速和所用扇区等方面严格的同步吗?为什么?
9. 什么是热备份磁盘?什么叫热插拔技术?
10. 阵列磁盘设备应用什么类型的接口卡?该接口卡上大体有些什么组成部件?阵列盘中的RAID0,RAID1,RAID4和RAID5指的是什么类型的容错功能?

第10章 输入输出设备

本章和下一章的教学内容，将围绕计算机输入输出系统的组成、功能、运行方式、具体使用方法等为主线索来进行组织。计算机输入输出系统，通常由计算机总线、输入输出接口和输入输出设备3个层次的逻辑部件和设备共同组成，本章主要讲解有关输入输出设备的内容。有关计算机总线、输入输出接口以及输入输出方式等方面的知识安排到下一章讲解。

输入输出设备又称计算机的外围设备，是有一定操作功能的比较完整、相对独立的精密机械电子装置，用于完成面向计算机操作人员的输入输出功能。当前的计算机系统中，输入输出设备的种类繁多，功能多样，组成和运行原理各不相同，本章只选择最常用、最基本的典型设备进行简要讲解。在输出设备中，属于显示器设备的，重点介绍阴极射线管显示器、液晶显示器；属于打印机设备的，主要讲解针式打印机、喷墨式打印机和激光印字机。在输入设备中，只是最简单地介绍一下计算机键盘和鼠标器的运行原理。本章的内容，一般叙述多于深入的原理性知识，有意识地避开设备中涉及的精密机械、光学、流体力学、微波技术等知识。

10.1 输入输出设备概述

计算机应用的普及，使越来越多的电子设备成为计算机的输入输出设备。早期的计算机只有纸带、穿孔机等外部设备，而现在以键盘、显示器或者它们的结合即终端已经成为计算机的标准输入输出设备，而硬盘、光盘等也成为必备的外部存储设备。在家用计算机领域，多媒体设备，如DVD、手写板、语音输入和数码相机也逐渐进入了计算机外部设备的大家庭。总之，计算机外部设备正向着种类繁多、功能丰富、智能化的方向发展，在计算机组成中的重要性也逐渐提高。图10.1是一个简单的输入输出设备列表。

- 输入设备
  - 键盘
  - 图形输入设备：鼠标、图形板、手写板
  - 图像输入设备：扫描仪、传真机、数码相机
  - 条形码阅读器
- 输出设备
  - 显示器(字符、图形、图像)
  - 打印机(针式、喷墨、激光)
  - 绘图仪

图10.1 输入输出设备列表

10.2 常用的输入设备

1. 键盘

计算机键盘是用于手工向计算机送入操作命令、源程序语句、运行程序所使用的数据等内容的输入设备,应用非常普遍。计算机键盘由机械部分和电子线路部分组成,并通过串行接口与计算机主机连接,向CPU送入所敲击按键的编码。

它的机械部分的组成,从外形上看,是在长方形的部件上以横行竖列位置关系依此排放的许多按键,每个按键上用字母、文字或符号标明这个按键的含义和作用。这些按键中,除了字母、数字、标点符号、数学符号等基本输入按键之外,还包括一些编辑功能键和操作控制键(含复合用按键)。从内部看,每个按键都是用小弹簧支撑着,按键按下去之后会把按键上导电件与其下面金属件接触上(称为按键闭合,实现电信号连通),松开手之后,小弹簧把按键顶起,使按键与其下面金属件脱离接触(按键松开,实现电信号断开),由此看来,按键相当于一个机械开关。

它的电路部分的组成,全部做在一块印制电路板上。其主要功能,一是识别按下的是哪一个按键,并产生出该按键对应的编码信息;二是把这一编码从并行格式转换成串行格式,逐位传送给计算机主机。这些功能是用板上的一个专用的CPU芯片控制完成的。

识别闭合键,多采用行扫描法完成,即找出闭合按键在按行列关系布置中所处的位置。具体办法是,给出一个n行×m列的逻辑电路,并把键盘上的每个键分配在这个n行×m列的交叉点位置,如图10.2所示。运行时,轮流为n行中的一行接通低电位,其他行给高电位,并检查m列的各列的电位值。若所有这些按键都未按下,都处于断开状态,则每一列都不会与行连通,不管行上有高或低电位,所有列的电位都为高,是5V电源经一个电阻给出的。若只有一个键按下,即进入闭合状态,就会把它所连接的行线和位线连通,若该行线上为低电位,则使位线也为低电位(5V电源的电压都降在电阻上),该位线上的低电位与其他行线的高电位被相应的二极管断开,其他位线仍为高电位。这就是说,通过一个行线和列线同时为低电位(其他所有的行线和列线都为高电位)来表明一个闭合键的位置。再用行线和位线的编码去查表,就得到闭合按键的编码值。在具体实现中,对行线和位线的控制是应用并行接口电路完成的。把并行接口的输出口接到行线,由CPU轮流送出只有一位为0值、其他位均为1值的信号,而把列线接到并行接口的输入口供CPU来读,CPU分析读入的内容以判断列线的电位情况。请注意,这里说的CPU是指键盘线路板上的CPU,而不是计算

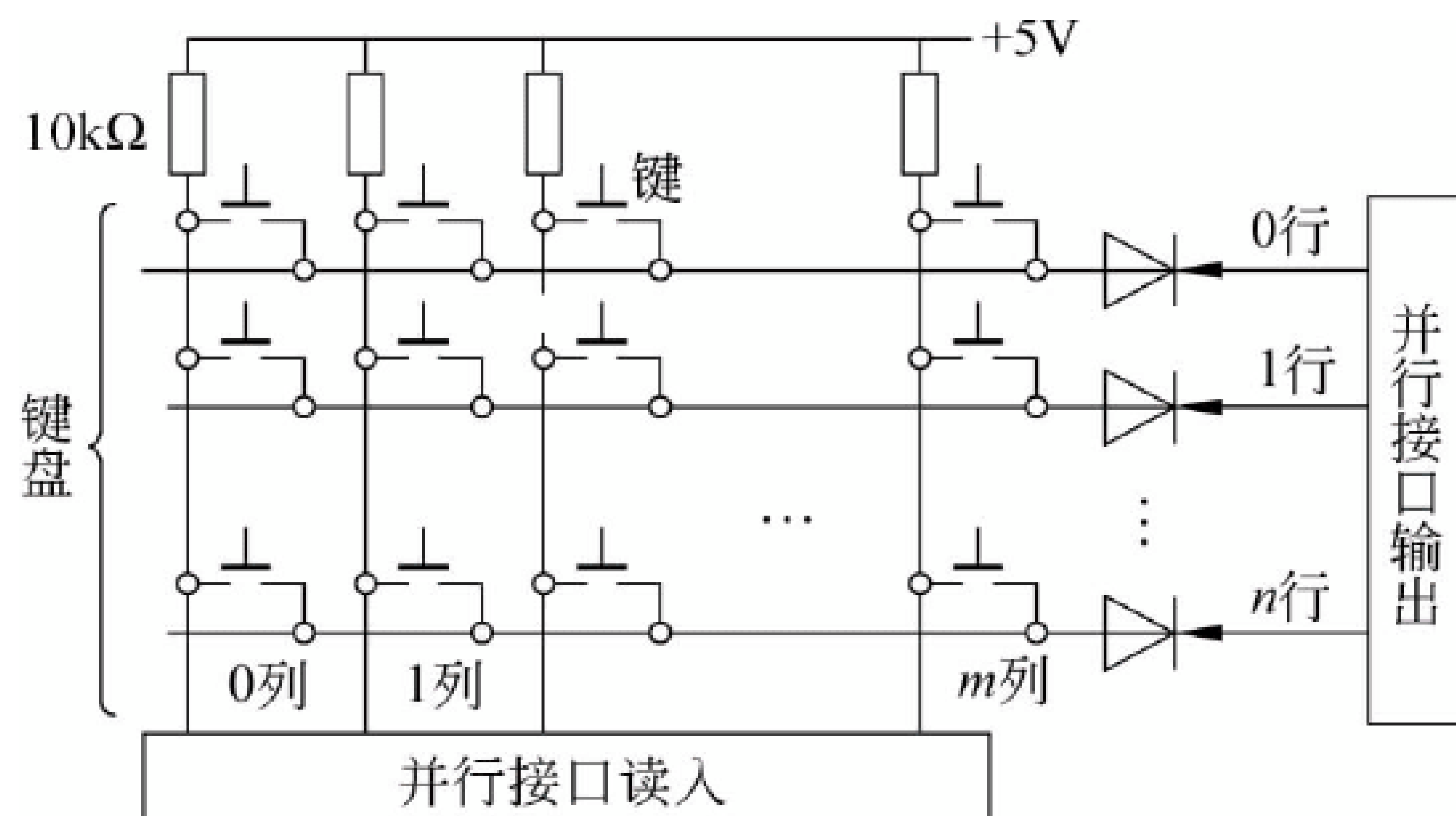


图10.2 键盘中识别闭合键的逻辑线路

机主机的CPU。同样,用行线和位线的编码去查表,以得到闭合按键的编码值的操作也是由键盘线路板上的CPU完成的。

键盘的具体应用中,还应解决抖动和组合按键等问题。抖动要区分的主要是键被按下一次还是多次,组合键要用于判断是合法组合还是非法组合,合法组合(如Shift键和字符键)要能正确响应,非法组合要能给出出错信息。

键盘通常都通过串行接口与计算机主机连接,把敲击的键的编码逐位送到主机一端的串行接口中,由主机一端的串行接口完成串行格式到并行格式的转换,供主机CPU读取。

2. 鼠标

鼠标是随着图形用户界面出现的输入设备。图形界面的出现使计算机的易用性大为增强,也使更多的对计算机工作原理一无所知的人成为计算机用户,他们希望能有在图形界面上进行点击的设备,这就是鼠标产生的原因。

从感应方式看,主要有3种类型的鼠标:机械式、光学式和光电式。机械式鼠标内部有两个轴互相垂直的小轮,底部有一个橡胶球。当鼠标移动时橡胶球进行旋转,通过摩擦带动两个小轮进行转动,每个轮子各驱动一个可变电阻。通过测量电阻值的变化,就可以得到鼠标在每个方向上的位移。

光学式鼠标底部没有轮子,也没有球,而是由一个发光二极管和一个光检测器代替。光学鼠标下面必须是一个特制的鼠标垫,鼠标垫上是距离很近的线条组成的矩形格。随着鼠标在垫子上移动,光检测器根据从LED反射回的光线总量的变化来感知鼠标越过了线条,鼠标内部的逻辑电路可以计算出每个方向上移动的格数,计算鼠标的位移。

光电式鼠标和新式的机械式鼠标类似,底部是一个可以绕互为垂直的两个轴旋转的小球。两个轴上连有译码器,上面有光线可以通过的小裂缝。鼠标移动时,带动轴旋转,当裂缝正好位于LED和检测器之间时,检测器可以感知到一个光脉冲。脉冲数和鼠标的位移成正比,对脉冲计数后就可以得到鼠标的位移。

虽然对位移的感知方式各不相同,但每种鼠标在移动一个最小位移单位后,都将向计算机发送一个3字节的串。一般情况下,这个串通过串行接口进入计算机。第一个字节是一个有符号整数,表示鼠标在最后的100ms在x方向上的移动量;第二个字节与其类似,表示的是y方向的位移量;第三个字节描述鼠标当前键的状态。有时,两个坐标方向的位移量各需要用2个字节表示。

当这些字节送到计算机中后,计算机的底层软件接收这些信息,并将相对位移量转换成鼠标的绝对位置,然后在屏幕上对应位置显示一个箭头来表示鼠标的当前位置。当箭头指向正确的菜单项时,如果用户按下鼠标上的键,计算机就可以从箭头在屏幕上的位置计算出哪个菜单项已被用户选择,然后执行用户所希望完成的工作。

10.3 常用的输出设备

10.3.1 点阵式输出设备基本原理

计算机程序的运行结果和中间过程,都需要输出设备提供给用户。最常用的输出设备有显示器和打印机。这两类输出设备基本上都是以点阵方式运行的,通常都是期望把某些

信息,例如字形、图形、图像等,以计算机用户可见的某种形式表示出来。在计算机显示器屏幕上,这些被显示的内容,是以可见光形式表现出来的;而在打印纸上,通常是以“印刷”(染色)的效果表现出来的。它们共同的特点是,要表示的信息,最终要以平面上的各种可见的“形状”体现出来,而这些“形状”,不管其简单还是复杂,在原理上又都是以许多断续的点的不同布局表示出来的,当一些点彼此之间靠得很近的时候,使人看上去就好像连接在一起的样子。

以点阵方式运行的设备重点解决的问题是要设法找出组成各种形状的点的布局规律,以及在有关的输入输出设备中,如何针对这些规律把这些点显现出来。从组成各种形状的点的布局规律来看,大体有两种情况。第一种情况是,每个被表示的对象有确定的形状,如中、西文字符,标点符号,数学运算符号等,更进一步说,一些简单的几何形状也可以用符合某种规律布局的一些点表示出来,如直线、圆、矩形等。在需要表示这样一些对象(呈现它们的形状)时,例如要表示字符,就可以事先用某种办法把这些字符的点的布局设计保存在存储器中,需要时再把它们复现出来,这是在字符类型设备中常用的办法。另一种情况是,组成被表示对象的点的布局没有确定的规律可言,例如一幅图像到了计算机内也得被表示成由许多点组成的方阵格式,图像的内容就变成在一个确定的平面范围内通过摆放上许多相应的点来加以表示,此时这个方阵中哪些位置上有点,哪些位置上无点,完全是由图像本身的内容决定的。这些点布局规律的不确定性决定了在具体的处理中需要记忆该画面上对应每一个点的位置上有无点的实际情形。问题处理的焦点,就变成如何把图像上所表现的连续变化的内容,在计算机设备的指定的平面上用离散的点表示出来,如何表示的更真实准确,这正是图像处理领域要解决的问题。

前面的讨论只限于用点有无的形式来表示“形状”,属于最简单的应用方式。很容易想到,为每一个点分配一个二进制的位(bit)就可以区别出点的有无。例如,用该 bit 的值为 1 表示相应位置上有一个点,该 bit 的值为 0 表示相应位置上没有点。这样,一个平面范围内的全部点的布局情形,就可以用一长串二进制数位组成的数据结构来表示,或许将这一长串二进制数位,组织成一个二维数组的形式,以方便找出二维数组中的一个二进制数位与平面中一个具体位置的对应关系。

在实际应用中往往还必须解决其他方面的许多使用要求,才能使上述思路达到更高的应用标准。例如,对这些点安排得多密才更为合理,显而易见,安排更密的点可以把要表示的形状呈现得更精细准确,但表示同样大小的形状用到的数据量会更多,对输入输出设备的处理精度要求也越高。此外,还需要找出记忆与处理一幅图各处不同的亮暗层次或不同颜色的办法,对字符也有用不同深浅程度或不同颜色表示的使用要求。对显示器而言,在使用单一颜色的情况下,只能用各处的不同亮度(深浅程度,通常被称为灰度级)来体现一幅图的层次感,即每个点可以用几种不同的亮度显示。此时,为表示一个点,就不能只用一个二进制位,而要用多个二进制位,例如 b 位,具体位数 b 取决于显示的亮度等级数 S,其关系是 $S=2^b$,亮度等级越多,用于表示一幅同样大小的图的数据量也就越大。在使用多种颜色的情况下,不同的颜色是用规定的 3 种基本颜色按不同的比例关系混合而呈现出来的,对彩色显示器或彩色的打印机都是如此。为此,要求每个点都可以呈现不同的颜色,为表示 256 种颜色,要用 8 个二进制位,若希望对每种基本颜色都有 256 种亮度,则表示每一个点要用 24 个二进制位,即 3 个字节(B),每一个点就可以有 $256\times256\times256$ 种颜色,通常被称为无限

种颜色,也被称为真彩色。此时,为了表示一幅由 1024×768 个点组成的图,需要使用 1024×768×3 个字节的数据量。

上述的内容,只是点阵式设备的基本运行原理中的简单概念,要把它在真正的设备中实现出来,还有许多技术方面的问题要具体解决。首先把这些设备中用到的属于共同性的知识说明清楚,不但可以减少介绍不同设备运行原理过程中的重复性叙述,而且对深入理解这些知识也大有好处。

10.3.2 显示器的组成和运行原理

显示器设备是以可见光形式显示信息的输出设备。当前使用最多的是以阴极射线管(Cathode Ray Tube,CRT)为主体的显示器,其次是液晶显示器(Liquid Crystal Display,LCD)。从它们显示的内容区分,有以显示字符为主的字符显示器,通常也兼有显示一般质量的图形、图像的功能,也有以显示高质量的图形为主的图形显示器,二者的复杂程度和价格相差较大。目前的计算机系统中,通常使用的是一般的显示器,又称监视器(Monitor),结构上大体是一台没有高频放大部分的电视机,本身还算不上一台完整、独立的设备,因为它的控制器部分被分离出来,并且用插接在主机中的一块显示卡实现,通过显示卡向显示器传送视频信号。

显示器设备是属于以点阵方式运行的典型设备。显示的内容以可见光形式呈现在显示器的显示屏幕上。显示屏幕是一个矩形的平面装置,它的大小习惯上用其对角线的长度表示,使用英寸为长度单位,常用的有 15 英寸、17 英寸、19 英寸、21 英寸等不同尺寸。为实现显示,沿水平和垂直两个方向把屏幕分成许多小的区域,一个小的区域对应一个发光点,每个发光点称为一个像素,一个屏幕上所提供的全部像素的数目被称为分辨率,常用的有 800×600、1024×768、1280×1024、1600×1200 等多种分辨率,它与屏幕的尺寸和像素之间的距离有关,像素之间的距离多为 0.31mm 或 0.28mm。显示器有单色和彩色两种,在单色显示器中,是用所显示内容的亮暗程度,即灰度级(Gray Level),来表现显示内容的层次感;彩色显示器则可以用比较真实的颜色来显示一个对象的形状和颜色,一台彩色显示器所能提供的颜色种类,通常可以在系统中进行设置,颜色种类越多,表现力更强,但为表示同一个对象所占用的存储器容量要更大。

1. 阴极射线管显示器

阴极射线管显示器是目前使用最广泛的显示器件,最早用于电视接收机,然后用于计算机系统,作为字符、图形和图像显示器。阴极射线管是一个漏斗形的电真空器件,由显示屏、电子枪和偏转控制装置 3 部分组成,如图 10.3 所示。

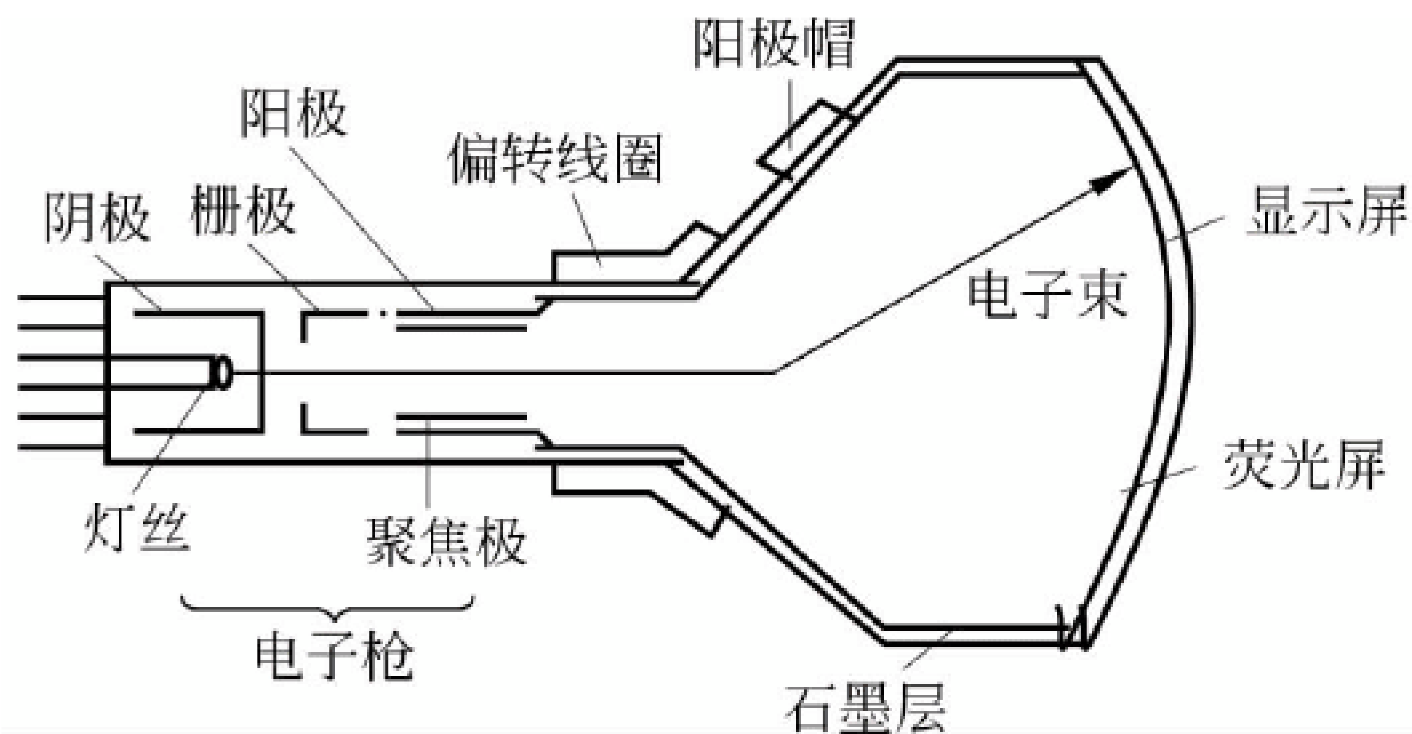


图 10.3 阴极射线管的组成

显示屏是显示信息的主体部分，由玻璃屏和涂在其内壁的荧光粉薄层构成，这层荧光粉可在电子束撞击下发出不同颜色和亮度的光点。为了在显示屏上显示信息，必须有为其提供电子束和选择电子束在屏幕上撞击位置的相关部件。

电子枪是用于产生电子束的部件，由灯丝、阴极、栅极、阳极、聚焦极几部分组成。灯丝在通电之后产生热量，使阴极被加热，变热的阴极会释放出大量的电子；栅极用于控制这些电子通过栅极进入阳极区域，进而撞向显示屏的电子的数量，即打向显示屏的电子束的强弱；阳极实现对电子束的加速，确保电子束有足够的动能，以提高显示屏的显示亮度；聚焦极用于对电子束进行聚焦，把原来初速不等、方向不尽相同的电子聚焦成很细的一个电子束，以便打到显示屏上能形成一个很小的亮点，保证较高的显示清晰度。

偏转控制装置是指套在阴极射线管尾部的偏转线圈，用于控制电子束沿着水平和垂直两个方向的运动轨迹，以便准确地控制一束电子能打到显示屏幕上任何一个位置，这是在显示屏幕上全屏显示信息所必须实现的控制功能。

对彩色显示器，显示的颜色应由红、绿、蓝 3 种基本颜色按一定比例关系搭配而成。为此，对显示屏上的每一个像素，都要由能在电子束照射下发出红、绿、蓝 3 种颜色的 3 个小荧光粉点组成，可以把它们排列成正三角形状，再为它们各配备一个独立控制电子束强度的电子枪，并确保 3 个电子枪发出的电子束能准确地打在各自对应的小荧光粉点上。为此，3 个电子枪也要排列成正三角形状，并在荧光屏附近安装一个布满小孔的荫罩板，其小孔数与 3 色荧光粉点的组数（单色时的像素数）一致，以确保 3 个电子枪发出的电子束能穿过同一小孔分别打在各自对应的小荧光粉点上。

电子束在显示屏上按某种轨迹运动被称为扫描，控制电子束扫描轨迹的电路被称为扫描控制逻辑部件，常用的扫描方式有光栅扫描（Raster Scan）和随机扫描（Random Scan）两种，二者的性能和价格差异较大。在光栅扫描方式下，电子束要从左到右、从上到下扫描整个屏幕，扫描控制本身不必区分什么位置上有点要显示，什么位置上无点不显示，它只是控制电子束在整个屏幕上重复移动，显示的具体内容则通过另外的逻辑线路提供。在这一扫描方式下，有逐行扫描和隔行扫描两种方案，逐行扫描是从屏幕顶端开始，依次连续扫描所有各行，隔行扫描是这次只扫描行号为奇数的全部各行，下次再扫描行号为偶数的全部各行。电视中普遍采用的是隔行扫描技术。由于光栅扫描与电视系统使用相同的技术，技术成熟性好，产品价格便宜，被广泛地用在计算机的显示器中。它的缺点是扫描冗余时间多，分辨率较低，故主要用于普及型的字符显示器。在随机扫描方式下，电子束只扫描在屏幕上有显示内容的位置，而不是整个屏幕，所以这种扫描方式画图速度快，分辨率高，故主要用于高质量的图形显示器。其缺点是，它的扫描控制逻辑比较专用、复杂，产品生产批量不够大，价格较高。

电子束打在荧光粉上发出的光所持续的时间，被称为余辉时间。余辉时间的长短，主要取决于荧光粉材料的特性，显示器中通常使用余辉时间比较短的这一种。为了在屏幕上有稳定的、至少人们看上去无明显闪烁感的画面，就要把显示的内容不断重复显示，一般来讲 1s 要重复扫描整个显示屏幕 60～100 次，这正是扫描控制逻辑应完成的控制功能。

2. 液晶显示器

CRT 显示器经过长期的发展，技术成熟，成像质量也高，使用十分普及。但由于其体积大，耗电量高，显然不适合移动计算环境。目前，在笔记本计算机上得到广泛应用的是液晶

显示器,而且,随着液晶技术的发展,成像质量不断提高,加之液晶显示器低能耗、低辐射的特点,使它大有与 CRT 显示器抗衡的趋势。

液晶是一种胶状的有机分子,可以像液体一样流动,但又有像水晶一样的空间结构,是 1888 年由一位澳大利亚的植物学家发现的,并在 19 世纪 60 年代被首次用来做显示器(如计算器、手表)。当液晶的所有分子都朝一个方向排列起来,它的光学性质将取决于光进入的方向和光的偏振性。使用特定的电场,可将液晶分子重新排列,也就可以改变其光学性质。更为独特的是,用光照射液晶,透射光的强度可以用电场控制。液晶的这种属性可以用来开发平面显示器。

图 10.4 给出了 LCD 显示器原理的示意图。图 10.4(a)给出了 LCD 的一般构造,LCD 显示器的屏幕由两块平行的玻璃中间夹着一层密封的液晶组成,两块玻璃分别连着透明的电极。后面那块玻璃之后有一束光线(自然光或人造光)照射到屏幕上,连在玻璃上的透明电极用来在液晶中产生电场,用不同的电压加在屏幕不同的位置,控制显示的图形。粘在前后两块玻璃板上的是人造偏光板,因为液晶显示技术要求使用偏振光。

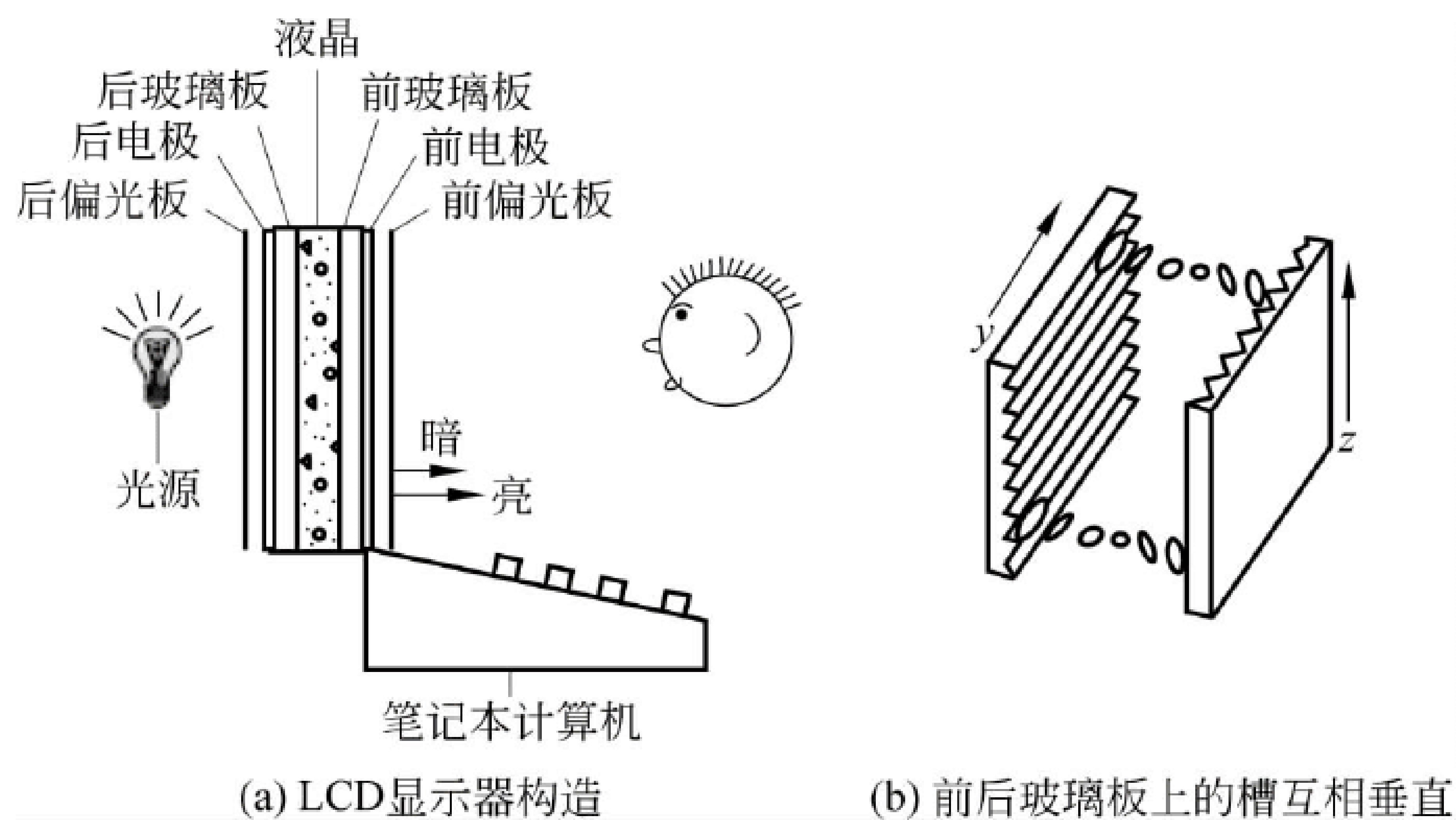


图 10.4　液晶显示器原理示意图

虽然目前有多种 LCD 显示器,但我们还是以绞合向列型(Twisted Nematic,TN)显示器为例进行介绍。这种类型的显示器后玻璃板上有许多微小的水平槽,前玻璃板上有一些微小的竖直槽,如图 10.4(b)所示。没有电场存在时,液晶分子将顺着槽排列。由于前后玻璃板的槽互相垂直,所以液晶分子(同时导致晶体结构)从后到前将从水平到竖直变化。

显示器后面是一块水平偏振板,只允许水平偏振光通过。显示器前面是垂直偏振板,只允许垂直偏振光通过。如果中间没有液晶的话,从后面进来的水平偏振光将被前面的垂直偏振板完全阻碍,使屏幕一片漆黑。

但是,有处于中间的液晶分子进行引导,光线将转变其极性,完全成为垂直偏振光。这样,如果没有电场控制,LCD 显示器上也将是一片光亮。通过在选定的位置加上一定的电压,液晶的结构将被破坏,阻碍那个位置的光线通过,将使该位置变黑。

一般有两种方式提供电压。在廉价的**被动矩阵显示器**上,两个电极上都是平行的导线。例如,在 640×480 的显示器上,后面的电极上可能是 640 根垂直的导线,前面也许是 480 根水平的。通过在某根垂直导线上加上电压后,再在某根水平导线上来一个脉冲,两根导线交叉点的电压将被改变,使该点短暂变黑。在下个点和下下个点上重复这个脉冲,一条黑线就

画出来了，与CRT的工作原理类似。正常情况下，整个屏幕将在1s内重画60次或者更多次，使眼睛觉得屏幕上有一个固定的图形，这也和CRT的工作方式一样。

另一种广泛使用的方式是**主动矩阵显示器**。它相对来说昂贵一些，但成像质量也要好一些。它不但有两组互相垂直的导线，在其中的一个电极的每个像素位置上还有一个微型开关。通过打开或关闭这些开关，可以在屏幕上产生跃变电压，也就在屏幕上产生跃变点阵。这些微型光管被称为**薄膜晶体管**（Thin Film Transistor，TFT），采用这种技术的平面显示器也常被称为**TFT显示器**。目前，大多数笔记本计算机以及许多桌面计算机配置的独立的平面显示器使用的都是TFT技术。

上面我们简单描述了单色液晶显示器的工作原理。可以说彩色液晶显示器在原理上和单色液晶显示器基本相同，但细节上要复杂得多。彩色液晶显示器中在屏幕上的每个点阵用光过滤器将白光分解成红、绿、蓝三原色，并使它们能独立显示出来，通过它们的线性组合，就可以创造出屏幕上的万紫千红的各种颜色。

CRT显示器和TFT显示器每秒钟都必须刷新60～100次，以获得稳定的显示质量。这些用来刷新的数据被存放在显示控制卡上的专用存储器——**显示存储器**（Video RAM）中。显示存储器中存放有代表一屏或多屏显示的位图数据。例如，对一个有1600×1200个**像素**的屏幕，显存中应存放1600×1200个数值，每个数值对应一个像素。实际上，显存中存放着好几个这样的位图数据，以实现不同屏幕图像的快速切换。

高端显示器中，每个像素被表示为3字节的RGB值，每个字节分别用来表示该像素点的红、绿、蓝颜色的亮度。根据物理规律，任何颜色都可以通过红、绿、蓝3种光进行线性组合获得。对于1600×1200分辨率的显示器来说，如果每个像素点用3字节表示，则其显存需要接近5.5MB的容量来存放图像信息，同时，对图像的任何动作都需要花费CPU相当多的时间。位图显示器对带宽要求也很高。为在1600×1200分辨率的显示器显示全屏幕、全彩色的多媒体图像，就要求在显示一帧图像的时间内，复制5.5MB的数据到显示存储器中。对全动画的图像，每秒要显示至少25帧，总的数据传输率要达到137.5MB/s。这种带宽要求远不是(E)ISA总线所能达到的，甚至也超过了最初的PCI总线的传输能力(127.2MB/s)，带宽始终是一个大问题。

为使CPU到显存的带宽得到提高，从Pentium Ⅱ开始，Intel增加了一条新的图形加速端口总线(Accelerated Graphics Port，AGP)来支持显存数据的传输。AGP总线以66MHz的频率传输32位的数据，带宽达到了252MB/s。为支持更高交互度的图形，后续的AGP总线版本以2、4，甚至是8倍速的速度，来提供足够的带宽，并不增加PCI主总线的负载。

10.3.3 打印机的组成和运行原理

打印机是最常用的输出设备，它把计算机输出的信息打印在纸上，可以较长时间保存。打印机多用RS-232串行接口或通用并行接口与计算机主机相连。随着USB接口在个人计算机中的普遍使用，现在的打印机基本上都支持USB接口。目前常用的打印机有针式打印机、喷墨式打印机、激光式打印机等多种，其中大多数只以一种颜色打印，但也有一些支持彩色打印，它们都属于以点阵方式执行输出功能的设备。

可以从不同的角度对打印机进行分类。从打印方式的角度，可以把打印机分成击打式和非击打式，击打式又被分为点阵针式和活字式两种。击打式是通过打印的机械装置撞击

色带以便把字形染印在纸上,活字打印机的打印的机械装置是多个刻有文字的小“锤”,类似于日常手工打字的英文打字机,它只适用于字符集很小的文字,如英文,要很好地打印中文汉字就有困难,更无法支持打印图形和图像;针式打印机的打印的机械装置是多个用电磁铁控制的打印针,通过组合这些针可以形成由点阵构成的随意的形状,故它可以用于打印多种语言的文字、图形和图像,通用性强。击打式打印机打印速度慢,噪声大,打印质量一般。非击打式打印机则是通过静电、喷墨等非机械撞击方式完成在纸上着色,打印速度快,噪声低,打印质量高,易于实现彩色打印,普及型的喷墨打印机和激光打印机得到广泛应用,高档的激光打印机在电子照排印刷系统中得到普遍应用。

1. 针式打印机

针式打印机属于点阵式打印机,价格低廉,功能可靠,但速度不高,噪声较大而且图形打印效果很差。目前主要有3种用途,第一是用来打印一些大张的表格;第二是适合打印一些小片的纸张,如现金收据、ATM或信用卡的交易记录以及登机牌等;第三是用在用碳蓝纸一次输出多份打印结果的场合。针式打印机由走纸机构、色带机构、打印头和一些逻辑电路等几部分组成,如图10.5(a)所示。

(1) 走纸机构由步进电机驱动走纸,有压轮摩擦走纸和链式纸孔走纸两种驱动方式,每打印完一行字符,走纸机构带动打印纸走一定距离。使用中,切勿让两种驱动方式(如果都有)同时起作用。链式纸孔走纸驱动方式下,打印纸不会走斜。

(2) 色带机构的作用是提供打印的色源,色带通常被安装在小圆盘形的盒内,在打印过程中,色带要在传动机构带动下不停地左右移动,以便使打印针比较均匀地撞击在整条色带的各个位置,减少对一个局部的磨损,延长色带的使用时间。

(3) 打印头是针式打印机中用于形成打印字符、图形的关键机构,通常有纵向排列的9个打印针或24个打印针两种类型。每个打印针装置的组成如图10.5(b)所示。每根打印针是由刚性、韧性都很好的金属材料制成,可以沿着导轨前后运动,运动的动力来自电磁衔铁的正向(前进)推动和机械弹簧的反向(后退)推动,当电磁铁的线圈中给出一个脉冲电流(对应需要打印一个点)时将产生磁场,电磁衔铁会在这一磁场作用下向前移动,推动打印针也向前移动并撞击色带;当线圈中的电流消失后,磁场消失,电磁衔铁也失去作用力,机械弹簧的反向推力把打印针推回原来位置。由于多个打印针可以同时被驱动,故一次打印是一个字符的一个纵向的点阵列,之后使打印头向右移动一个点的位置,就可以打印字符的下一个点阵列,几次之后就打印出一个完整的字符;再控制打印头向右移动一定间距,就可以开始下一个字符的打印过程,直到打印完该行上的全部字符。接下来是走纸机构控制走纸,打

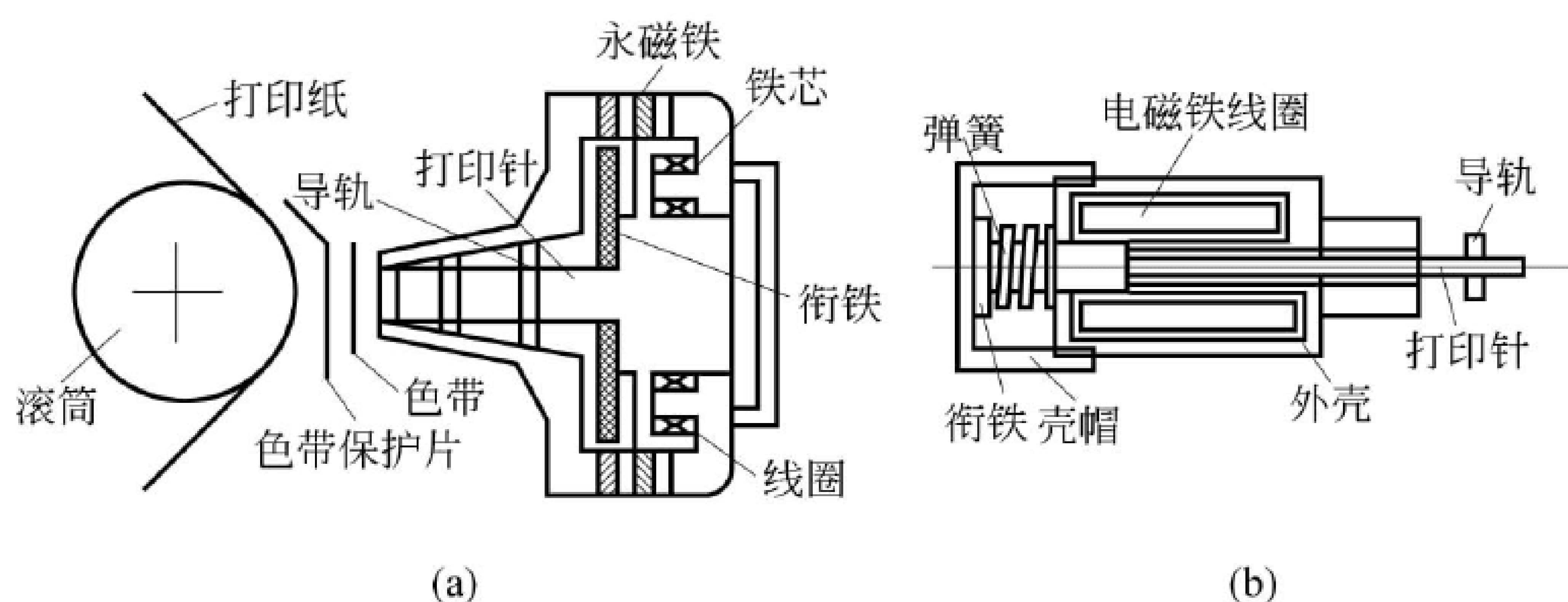


图10.5　针式打印机的组成结构

印头回到打印机的最左侧位置，开始下一行字符的打印过程。

(4) 针式打印机有自己的一些打印控制逻辑电路与缓冲存储器电路。打印控制逻辑中比较重要的是字形发生器，包括西文和中文字符的点阵信息库；缓冲存储器用于存储 CPU 送来的被打印的字符的编码，容量至少能存放一行字符的编码，现在的打印机通常可存放更多的内容；这样在 CPU 和打印机之间传送的信息量相对较少，由字符的编码找到它的点阵信息是在打印机控制逻辑电路之内完成的，并用这些点阵信息控制打印针的运动。当用针式打印机打印图形、图像时，CPU 就得直接提供图形、图像的点阵信息以控制打印针的运动。

针式打印机也可以完成彩色打印，它采用由 3 条不同颜色条组成的彩色色带，并可以在打印过程中通过上下移动色带，使不同颜色的色带条处在打印头之下，从而打印出不同颜色的字符。彩色针式打印机实用性较差，应用相对较少。

2. 喷墨打印机

喷墨打印机是通过把很小的墨水滴喷射到打印纸上形成打印点来完成打印输出功能的。讨论喷墨打印机时，问题主要集中在如何提供出很小的墨水滴，又如何加速墨水滴的喷射速度，如何准确控制墨水滴落到打印纸上的位置，如何处理墨水的循环流动和过滤。解决 4 个问题有许多种不同的方案，从而也就有了以不同原理工作的喷墨打印机。下面我们将介绍其中常用的一种喷墨打印机。图 10.6 示意性地给出了这种喷墨打印机的组成和打印过程。

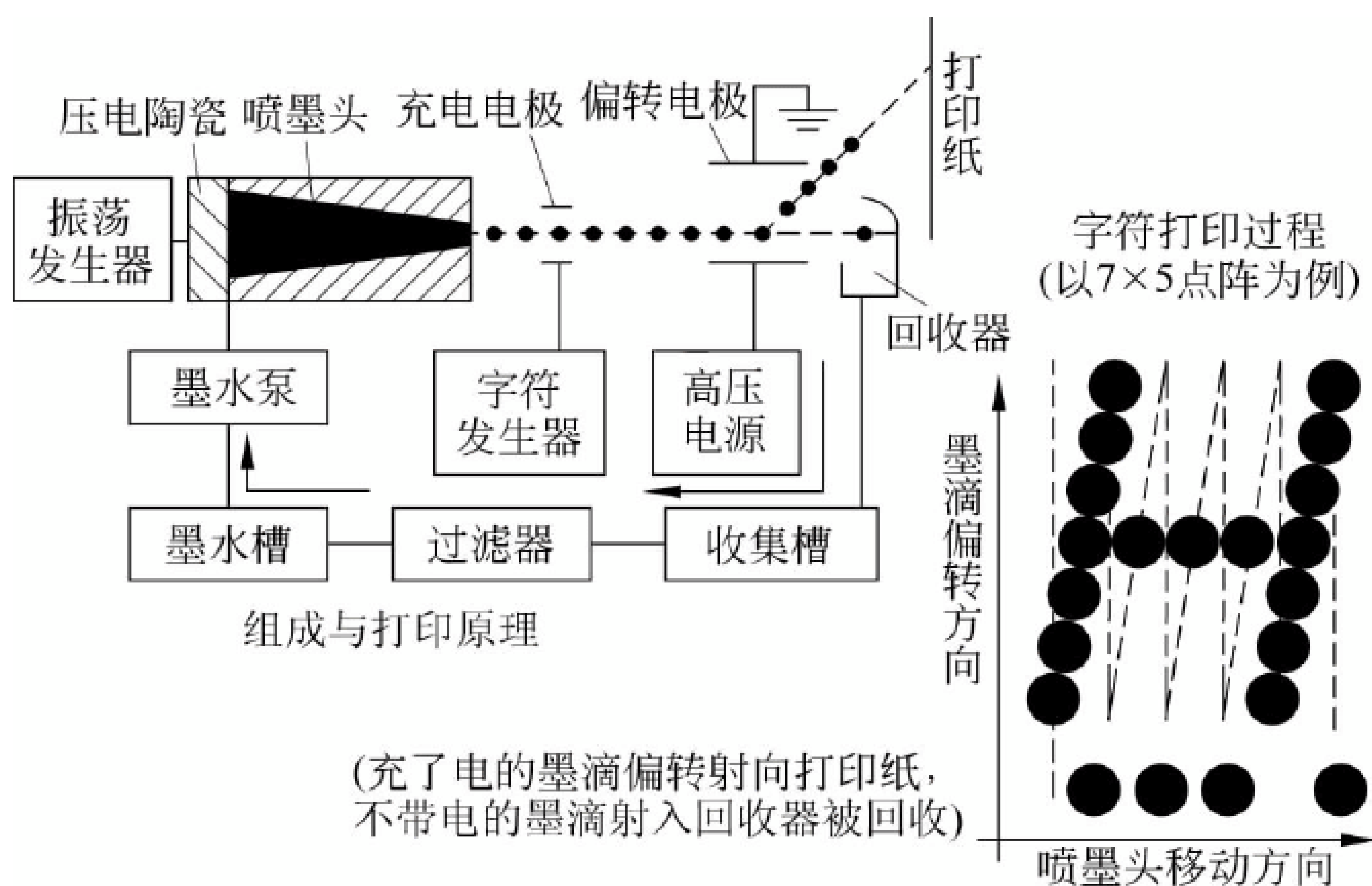


图 10.6 喷墨打印机组成结构和打印过程

在打印过程中，墨水滴是由喷墨头连续提供出来的。墨水在墨水泵的高压作用下进入喷墨头，可以通过喷嘴提供出一束极细的高速射流，射流的速度取决于墨水泵的压力。在喷墨头后部有压电陶瓷，它会在高频振荡器的作用下产生伸缩，从而可以使这束高速射流断裂成均匀的墨水滴射流。

喷墨头喷射出来的墨水滴进入充电电极区，通过在充电电极板上施加的静电场使墨水滴充电，要不要充电，充上多少电荷，由充电电极上所加电压决定，充电电极上所加电压又受字符发生器的控制。

充电电极之后是偏转电极板，其上接有恒定的高压，用于控制充上一定电荷量的墨水滴在垂直方向上的偏转距离，即墨水滴喷射到打印纸在垂直方向上的位置。水平方向的位置

控制是通过移动喷墨头完成的。墨水滴在垂直方向上的偏转距离,取决于墨水滴的充电量,充电量越多,偏转的距离就越大,不充电则不偏转,不偏转的墨水滴被回收槽挡住,不会射到打印纸上。充电电极直接在字符发生器的控制下,依据要打印的点在纵向点阵列中的位置,或该位置上没有打印点,确定向充电电极提供多高的电压或0电压,以决定为墨水滴的充电量,从而控制该墨水滴射向打印纸上确定的位置。

墨水循环流动系统由墨水泵、墨水容器、墨水过滤器、墨水回收器和管道等组成。其中的墨水泵,多级的粗、细过滤器,喷射嘴的正常工作是很重要的。

喷墨打印机也有单色打印与彩色打印两种类型。彩色打印是通过三基色原理,即分别喷射3种颜色,按一定的比例混合出所要求的彩色出来。这里介绍的只是基本原理,其实喷墨打印机设计中涉及许多领域的知识,有大量的分析计算工作,请有兴趣者参阅有关资料。由于喷墨打印机打印速度较快,打印质量较高,噪声很低,价格不高等优点而得到广泛应用,在支持彩色打印的设备中占有重要地位。

3. 激光打印机

激光打印机是激光技术和电子照相技术相结合的产物。它由走纸机构、激光扫描系统、电子照相部分和打印字形发生器与控制器等几部分组成。图10.7给出了激光打印机组成的示意表示。

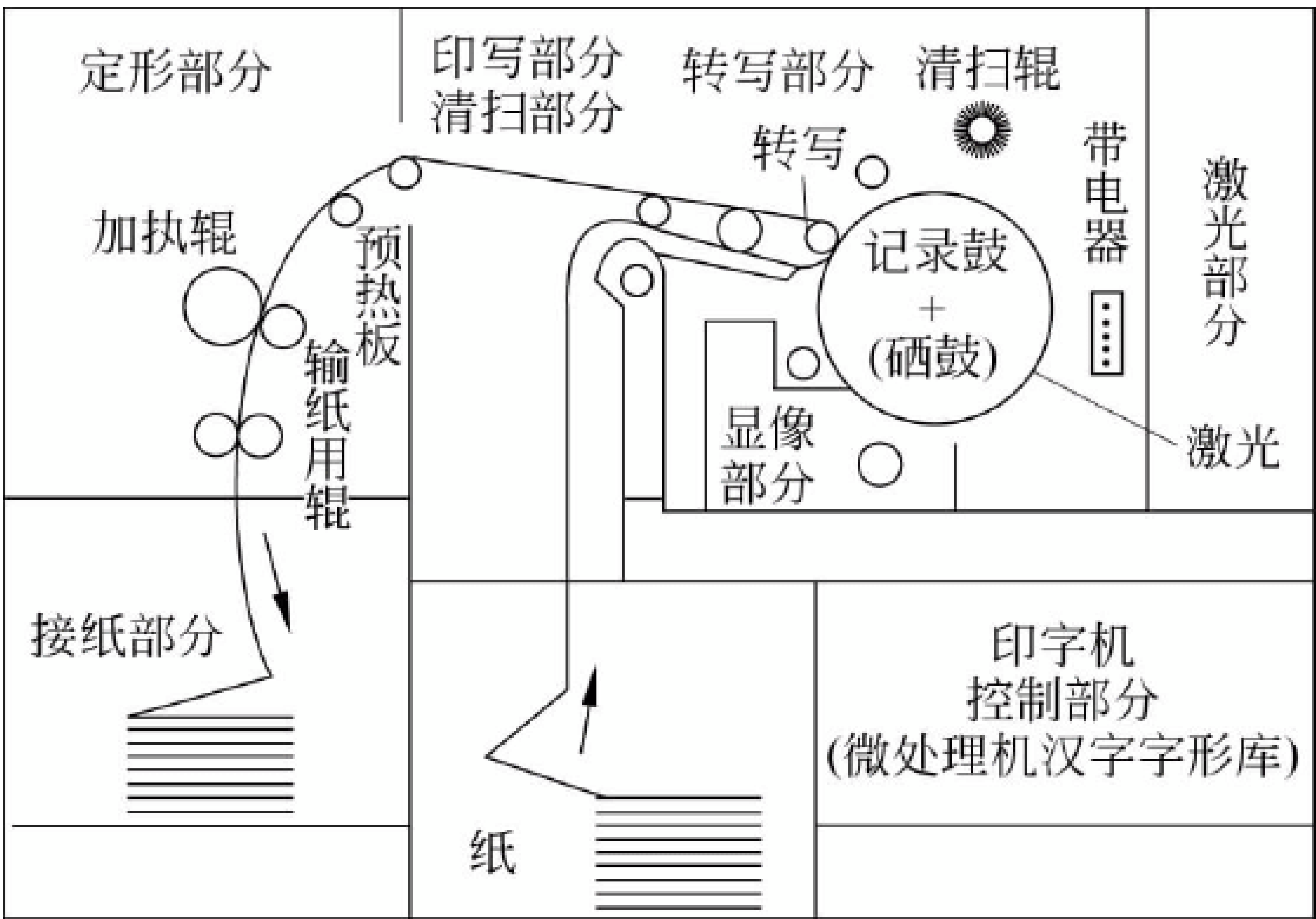


图10.7 激光打印机组成结构

激光扫描系统的功能是控制激光束能扫描到字形鼓(又称光导鼓,后面再讲)柱面的任何位置,它由激光器、偏转调制器、扫描器和光路系统组成。这些部分未在图10.7中加以表示。激光器大部分使用氦—氖气体激光器,它提供打印机运行所使用的光源。偏转调制器通常用声光器件(在器件内,用超声波改变媒体对光的衍射特性来改变光线的传播方向)调制激光束的传播方向,扫描器实现激光束沿字形鼓的轴线重复做横向移动,激光束的纵向移动是靠字形鼓的旋转完成的。这样,通过字形鼓的旋转和激光束的水平移动,就可以扫描到字形鼓柱面的任何位置。

电子照相部分的核心部件是字形鼓,这是一个圆柱形(鼓形)的物体,柱面高度光洁,镀有一层由硒—碲合金组成(P型光导材料)的具有良好光导特性(光线照射后电阻率降为原来的1/100到1/1000)的材料,用于完成对打印内容的照相、显影和转印。可对电子照相的全过程简述如下。

（1）准备阶段。开始时，通过电晕放电装置（用于使附近的空气电离）对光导鼓表面均匀地充上一层正电荷（离子化的空气分子），其表面电位可达几百伏，光导材料的内层感应出负电荷的电子，在没有光线照射的条件下，二者隔着光导层互相吸引，既不会中和也不会离去。

（2）照相阶段。在由被打印信息控制而提供出来的激光束扫描光导鼓时，光导鼓的不同部位就会发生不同的变化，在激光束照射到的部分（称为明区）电阻率降低，该处的电荷将会放掉，激光束未照射到的部分（称为暗区）的带电情况不变，这些剩下来的静电区域就是被打印信息的潜像。

（3）显影阶段。用的是墨粉和表面涂有树脂薄膜、直径为几百微米的玻璃珠为载体的混合物，运动中它们互相摩擦产生静电，墨粉被吸附在载体表面，当这些载体流过正经过这里的带有静电潜像信息的光导体表面时，载体表面上的墨粉被潜像的静电电荷所吸引，离开载体而黏附到了光导体表面，从而形成了由墨粉显示出来的字形。

（4）转印阶段。完成把光导体表面的字形墨粉转移到打印纸上，多数采用在打印纸的背面（打印纸的正面贴靠到光导体表面）通过电晕放出与墨粉所带电荷极性相反、电位更高的电荷，通过强力磁场把墨粉抢到打印纸上来，这就在打印纸上有了静电吸附着的墨粉字形。

（5）定影阶段。是把墨粉牢靠永久固定在打印纸上的工作。这是通过红外光加热或辐射加热的办法，用100℃的温度把墨粉熔化并凝沾在打印纸上，从而完成完整的打印过程。

激光打印机属于页式打印机，光导鼓每旋转一周打印一页内容。在开始下一页打印前，还要由清扫器清除光导鼓表面上剩余的墨粉，消电灯消除光导鼓上残存的电荷。

激光打印机打印速度快，打印质量高，噪声低，有普及型和各种高档型产品，被广泛应用在许多场所。

10.3.4 计算机终端

计算机终端是一台把键盘输入和屏幕输出功能集中在一起的、有自己控制器的独立完整的计算机输入和输出设备，通过串行接口与计算机主机连接。在计算机技术发展的早期，一些公司制造计算机，而另外一些公司生产终端。为使任何一台终端可以用在任何一台计算机上，电子行业协会（Electronics Industries Association，EIA）制定了RS-232C计算机—终端接口标准。任何一台支持RS-232C标准的终端都可以连到任何一台也支持这个标准的计算机上。RS-232C终端有标准的25针连线接口。它定义了接口的机械尺寸和形状、电平的高低和针上每个信号的含义。

计算机终端是大型机、巨型机时代的产物，虽然现在已经进入个人计算机普及的时代，但在机票预订、银行和其他面向大型机、巨型机的应用行业，这些终端设备的使用还是十分广泛，仍然发挥着巨大的作用。

本章内容小结和学习方法建议

输入输出设备种类繁多，功能多样，组成和运行原理各不相同，本章重点以点阵方式运行的设备的组成及其工作原理进行讲解，输入设备包括计算机键盘和鼠标器，输出设备包括

阴极射线管显示器、液晶显示器,针式打印机、喷墨式打印机和激光印字机,更多地围绕这些设备的电气功能实现来说明,而不是完整地介绍它们各个方面的知识。这里更多地强调了解几种最常用的输入输出设备的组成概貌和使用的问题,对于本章,同学们可以适当少花一点精力。

习题与思考题

1. 点阵式设备得以广泛应用的原因是什么?

2. 显示器的分辨率和灰度级与显示器的显示质量有什么关系?分辨率和灰度级的设置受哪些因素的限制?

3. CRT 显示器中为什么通常采用光栅扫描方式?光栅扫描与随机扫描各自的优缺点是什么?衡量显示器的质量有哪些指标?

4. CRT 彩色显示器比单色显示器有哪些更复杂的组成部分?

5. 液晶显示器的优点是什么?简述其发光原理和显示信息的过程。

6. 针式打印机由哪些部件组成?简述打印头的结构和打印出一个字符的原理性过程。

7. 提高针式打印机打印速度有哪些可行办法?针式打印机有什么优缺点?为什么很少用针式打印机进行彩色打印?

8. 喷墨式打印机的印字过程是更接近针式打印机的打印过程,还是更接近 CRT 显示器显示信息的显示过程?喷墨式打印机有什么优缺点?

9. 简述激光打印机打印一页文字的原理性工作过程。

10. 计算机的键盘是如何得到每个按键的编码的?

11. 计算机的终端和 PC 中的显示器在组成以及与计算机主机的连接关系方面哪些不同?

第11章 输入输出系统

计算机输入输出系统通常由计算机总线、输入输出接口和输入输出设备3个层次的逻辑部件和设备共同组成。有关输入输出设备的内容已经在第10章讲解过了，本章将主要讲解计算机总线、输入输出接口和输入输出方式这3部分知识。

总线用于连接计算机的各个部件为一体，构成完整的整机系统，在这些部件之间实现信息的相互沟通与传送；输入输出接口用于在计算机主机和输入输出设备之间实现正常连接和信息的传送配合；输入输出方式是指设备以什么样的方式和计算机主机进行数据交换。本章从实用的角度来讲解上述有关内容，包括必要的一些原理性知识和实例。

实例是为了深入地理解有关原理知识，并不要求同学们去强行记忆这些实例本身的具体组成细节、完成的具体功能、技术指标、具体使用方法等详细内容。把握住所学知识的主线索，掌握好学习原理性知识和实例的具体内容二者之间一个合理的精力分配，是正确、良好的学习方法的一个方面。

11.1 计算机输入输出系统概述

输入输出系统是完整计算机系统的重要组成部分，对计算机系统的运行性能有巨大影响。它的主要作用是连通计算机的各个功能部件和设备，在它们之间实现数据交换。输入输出系统的硬件部分主要由计算机总线和输入输出接口两部分组成，软件方面则需要有操作系统软件的支持。

输入输出系统也许是计算机系统中最复杂多变的部分，原因是多方面的。首先是有太多的CPU系列和型号，它们各自的运行速度、处理功能、接口逻辑都不相同；又有更多的外围设备，它们各自的运行原理、提供的功能、读写速度、接口逻辑更是千差万别；要把这么多不同的部件（设备）都能连接到一起，显然不是一件简单的事情，花样太多。其次，在计算机系统中，会有许多不同的使用要求，不同的人和不同的应用场合，对算题速度、输入输出速度、单位时间输入输出的数据量、对随时发生事件的响应与处理的速度、对系统总体性能的要求上各不相同，差异太大。企图用一种方式，一套办法全面解决这些问题显然是不现实的，至少从价格性能比的角度来看是不可接受的。因此，应该在系统配置的灵活性、良好的可扩展性、硬件与软件的合理配合等多方面来解决问题。

首先，众多的部件和设备要相互连接与交换信息，建立尽可能公用的交换信息的通路是

必要的,而且要提供各部件(设备)协调使用这些通路的规则。这些组成部分,在计算机内就是总线系统(Bus System),正如同在城市修筑的马路和建立的交通信号灯系统一样。总线结构、总线宽度、总线时钟、总线仲裁将是在计算机总线一节中要讨论的问题。

其次,要把众多不同的CPU与各种不同的输入输出设备连接起来,要求二者任何一方做出修改以适应对方都是不可接受的。最好的办法是在二者之间放置一个功能电路,把二者之间的连接、沟通、匹配、缓冲等种种需求都放到这个电路板中解决。由于该电路要解决的只是一个确定的CPU和一个确定的设备的对接,其复杂程度大为简化。这个电路板被称为设备接口。为连接多种CPU和多种设备,接口的种类数目最大的理论值是二者种类数目的乘积。在计算机接口一节将介绍通用可编程接口的一般组成,并以实例形式给出常用的串行接口的组成和使用知识,再简要说一下并行接口的有关内容。

最后,如何支持多个I/O设备并发执行输入输出操作,如何降低在输入输出操作过程中对CPU干预的需求。与CPU相比,许多设备的读写速度是非常慢的,如果要求CPU一定等待这些设备读写完成之后才开始执行下一条指令,CPU的大部分时间将花在等待上,系统性能会悲剧性的降低。为此,引进了程序中断方式和内存储器直接访问方式(DMA),甚至于另外配备一台小型计算机专门协助主CPU处理输入输出操作,以保证主CPU的更强的计算能力被用到更重要的处理中。这些是在常用的输入输出方式一节中要讲解的内容,这里重点讲解中断和DMA的概念,以及它们的请求、响应和处理过程。

11.2 计算机总线

11.2.1 总线概述

总线是计算机中多个设备共用的电子通道,一般可根据它的不同功能对它进行分类。它可用在CPU内部,为进出ALU的数据提供通道,例如内部总线;也可以用在CPU外部,将CPU和内存或输入输出设备连接在一起,例如系统总线和输入输出总线,每类总线都有各自的要求和特性。我们这里主要讨论系统总线和输入输出总线,也就是所谓的外部总线。

外部总线设计的初衷之一是为了使更多的外部设备能连接到总线上,和计算机内部部件进行通信,满足计算机的输入输出要求。因此,总线设计者必须详细定义总线的工作原则,并要求所有连接上来的设备都遵循,这些原则就是总线协议。另外,还要考虑总线的驱动能力,制定总线的机械和电子规格,使第三方的接口能够负载适宜,并对其提供合适的电压和时序信号等。

许多总线已经在现有的计算机上得到广泛的应用。其中比较出名的有Omnibus(PDP-8)、Unibus(PDP-11)、Multibus(8086)、IBM PC总线(PC/XT)、ISA总线(PC/AT)、EISA总线(80386)、Microchannel(PS/2)、PCI总线(多种个人计算机)、SCSI总线(多种个人计算机和工作站)、Nubus(Macintosh)、通用串行总线USB(现代个人计算机)、FireWire(前台收款机)、VME总线(物理实验设备)和Camac总线(高能物理设备)。这些总线中,有许多目前仍在使用中。出现如此多的总线标准,也是因为要接入计算机的设备实在是太多,又各具特点。

总线的作用是连接计算机的不同部件和设备。这些部件和设备有的是主动型的,能自

行对总线的数据传输进行初始化，也就是发起数据传输，我们把它们称为主设备，如CPU等；另外一些设备只能被动等待主设备的启动命令，我们称之为从设备，如主存储器等。当然，主设备和从设备也是相对的，某些设备就可以在某次数据传输中充当主设备，在另外一次数据传输中充当从设备。例如，当CPU要求磁盘控制器读写一块存储空间时，CPU为主设备而磁盘控制器为从设备。可是，当磁盘控制器要求内存接收它从磁盘驱动器上读到的字时，磁盘控制器就成了主设备。表11.1列出了几种典型的主从设备组合。任何情况下，内存都无法成为主设备。

表11.1　总线主从设备举例

主设备	从设备	举例
CPU	内存	取指令和数据
CPU	输入输出设备	初始化数据传输
CPU	协处理器	CPU提交指令给协处理器
输入输出设备	内存	DMA(直接存储访问)
协处理器	CPU	协处理器从CPU取操作数

计算机设备输出的二进制信号通常比较弱，无法驱动总线进行工作，尤其是总线比较长，或者上面的设备比较多时。因此，多数总线的主设备都要通过一片总线驱动器芯片和总线相连，该芯片实际上起了一个放大器的作用。与此类似，多数总线的从设备要通过总线接收器和总线相连。而对于那些既能做主设备，又能做从设备的设备，则通过总线转发器芯片和总线连接。这些总线接口芯片通常是三态门，在设备不需要和总线连接时可以使设备浮在总线上(逻辑上和总线断开)，而在需要时又能和总线连接，或采用与之类似的集电极开路方式和总线连接。当两个或两个以上的设备同时访问一根集电极开路线时，该线上的信号是所有这些设备的信号的逻辑或，我们称之为线或。对大多数总线，其部分信号线是三态信号，而另外的那些具备线或特性的信号线，是集电极开路信号。

一般来说，根据总线上传输的信号不同，总线可分为数据总线、地址总线和控制总线。数据总线在计算机部件之间传输数据(数据、指令)信息，它的时钟频率和宽度(位数)的乘积正比于它支持的最大的数据输入输出能力；地址总线在计算机部件之间传输地址(内存地址、I/O设备地址)信息，它的宽度(位数)决定了系统可以寻址的最大内存空间；控制总线给出总线周期类型、I/O操作完成的时刻、DMA周期、中断等有关的控制信号。

总线周期通常指的是通过总线完成一次内存读写操作或完成一次输入输出设备的读写操作所必需的时间。依据具体的操作性质，可以把一个总线周期区分为内存读周期、内存写周期、I/O读周期、I/O写周期4种类型。一个总线周期通常由两个时间段组成：地址时间(Address Time，CPU向内存或I/O设备送出地址信息到地址总线)、数据时间(Data Time，CPU完成数据读写)。若被读写的内存和外设的运行速度够快，可以保证在这一个数据时间内完成读写操作，则该总线周期在这一数据时间之后立即结束。若被读写的内存和外设的运行速度低，不能在这一个数据时间内完成读写操作，就必须再增加一到几个数据时间用于继续完成读写操作，之后才结束该总线周期。在增加出的这一到几个数据时间里，称总线处于等待状态，等待状态的存在无疑将降低系统的输入输出能力。

如果每次数据传输都要用两个时间(地址时间、数据时间)组成的完整的总线周期完成读写,则称这种总线周期为正常总线周期(Normal Bus Cycle),每次只能传输一个数据。若希望提高数据传输速度,也可以在给出一次地址信息(一个地址时间)后,接着用连续的多个数据时间依次传输多个数据,这种运行方式被称为总线的猝发传输方式(Burst Mode),又称成组数据传输方式,它需要CPU、总线、被读写对象各部件的支持。

11.2.2 总线结构

在计算机总线的发展过程中,出现了多种不同的总线结构,有简单的单总线结构,也有复杂的多总线结构,不同的总线结构具有不同的特点,但目的只有一个,就是要提高整个计算机系统的性能。下面我们先对单总线、双总线、三总线等不同结构的总线组成方案进行简单的介绍。

(1) 单总线是指计算机只使用唯一的一组总线,计算机系统中所有的部件、设备都连接到这组总线上。例如,美国DEC公司的PDP-11计算机就采用这一方案,称为Unibus。图11.1给出了其示意图。该方案的优点是结构简单,成本低,易于接入新的设备;缺点是不利于提高总线上的数据传输率,因为所有的部件、设备都连接到并争用这唯一的一组总线,每次只能在两个部件之间传输数据,其他部件是不能同时使用总线的,这被称为不支持总线的并发传输操作,不同的数据传输只能串行完成。请注意,这里说的一组总线,都是由地址总线、数据总线和控制总线3部分组成的(在双总线和三总线中亦是如此)。

(2) 双总线是指在计算机中配置两组总线,即在处理机总线上通过一块扩展总线的控制线路,提供出另外一组总线,称为输入输出总线。比较常用的有工业标准总线(ISA)和扩展的工业标准总线(EISA),主要用于连接一般的输入输出设备,它的总线时钟频率比较低(例如8.33MHz),数据线位数比较少(可以为1、2或4个字节);而处理机总线的性能则要高得多,例如PC系统中,多为33MHz的时钟频率和4～8个字节的数据线位数,可以实现CPU与主存储器之间的高速数据传输。这两组总线可以并发执行输入输出操作,使总线的输入输出能力和计算机系统的总体性能得到很大的提高。图11.2给出了双总线结构的示意图。

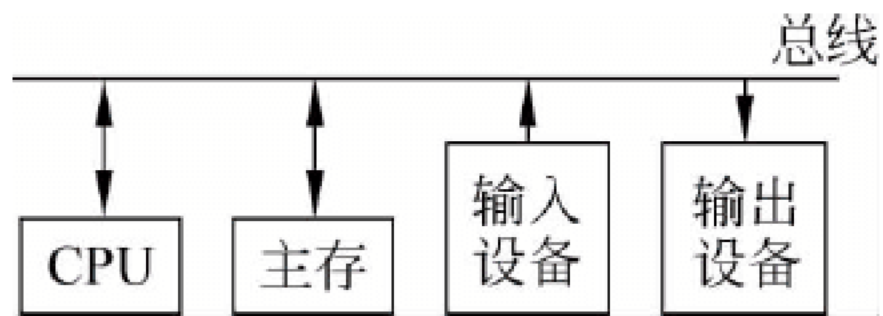


图11.1 单总线结构示意图

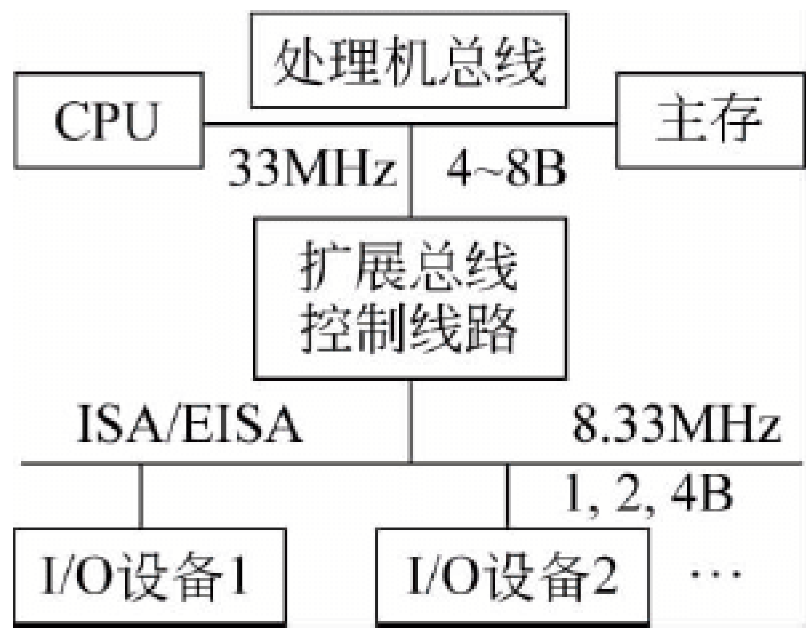


图11.2 双总线结构示意图

(3) 三总线是指在计算机中配置3组总线。即在处理机总线上通过一块被称为PCI桥的控制线路,提供出一组高性能的局部总线,称为PCI总线。而把原来的ISA总线和EISA总线从处理机总线上断开,并通过IO控制线路连接到这里的PCI总线上。把一些慢速的输入输出设备接到EISA(ISA)总线上。PCI总线的时钟频率比较高(例如33MHz),数据线位数比较多(例如4个字节),主要用于连接各种快速设备。而处理机总线的性能可能会更

高，例如66MHz或更高的时钟频率和8个字节的数据线位数。这3组总线可以并发执行输入输出操作，使总线的输入输出能力和计算机系统的总体性能再次得到更大的提高。图11.3给出了三总线结构的示意图。

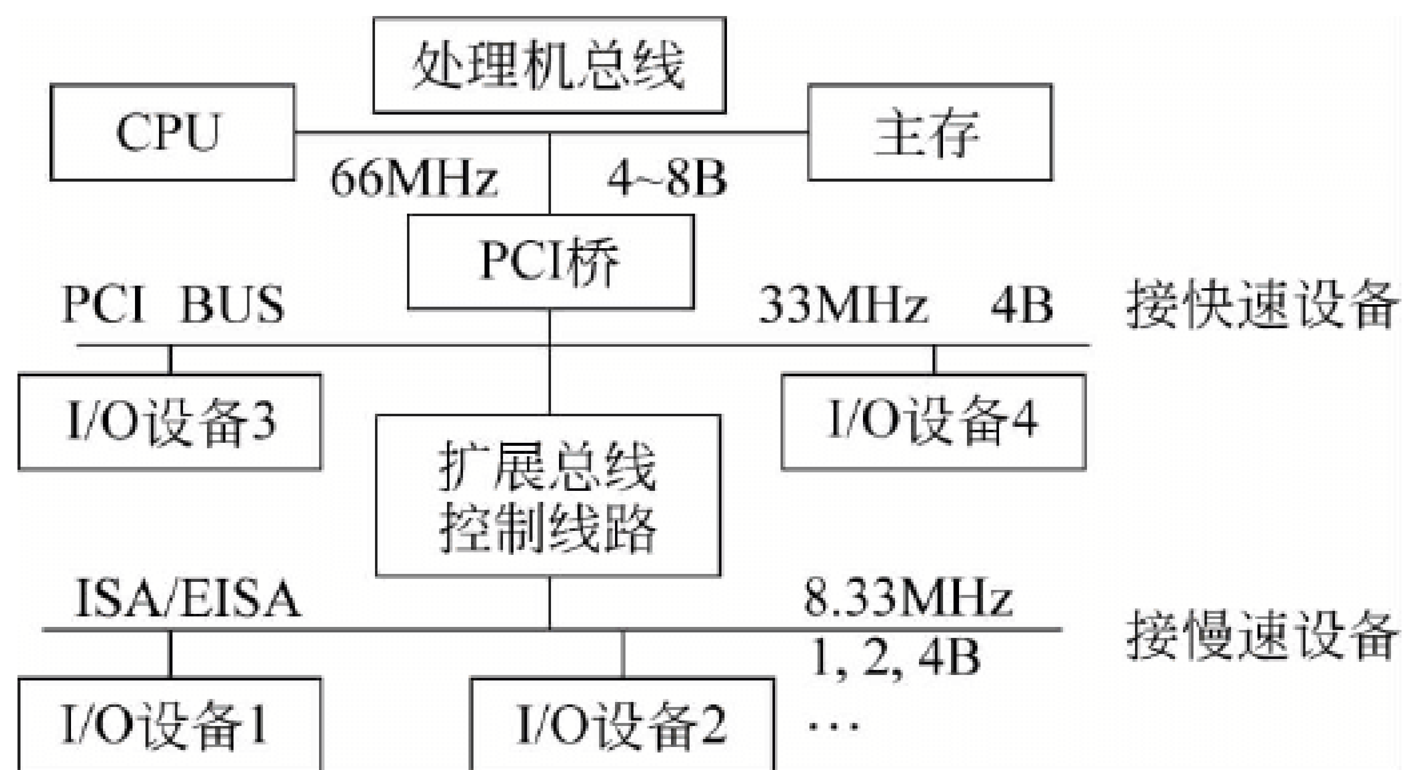


图11.3　三总线结构示意图

随着计算机技术的不断发展，总线的结构也在不断地变化。目前的个人计算机中的总线结构也不仅仅就是三总线结构这么简单，但总体上还是属于多总线结构。在本节后面我们将给出目前主流个人计算机系统的总线实例。

11.2.3　总线宽度

总线宽度似乎是总线设计中最简单的一个参数。总线中地址信号的根数越多，CPU能够直接寻址的内存空间越大。若总线中有n根地址线，则CPU能用它对2^n个不同的地址进行寻址。为达到更大的寻址空间，总线中需要的地址线就越多。对数据总线来说，情况也相似，需要一次传送的数据字长越长，数据总线的宽度也越宽。

问题是宽总线需要比窄总线更多的导线，同时也需要更大的物理空间(如主板上的总线要占用更多的面积)和更大的插头。这些因素都使总线的价格更昂贵。这就需要在寻址空间和系统成本之间进行权衡。带有64根地址线的总线，寻址2^{32}内存字节的系统要比只有32根地址线的总线系统成本高。

但是，从另一个方面看，计算机行业是一个高速发展的行业，用户对计算机提出的速度更快、存储容量更大的要求迫使计算机设计者不断对已有的总线进行位数的扩展。例如，早期的IBM PC，采用8088作为CPU时，有20位的地址总线，如图11.4(a)所示。这种PC可以寻址1MB内存空间。

80286出现后，Intel决定将寻址空间增加到16MB，这样就需要在总线中增加4根地址信号线(为保持向后兼容，不能影响以前的20根地址信号)，如图11.4(b)所示。这样，又不得不在总线中增加控制信号来处理新增的地址线。到80386时，又要增加8根地址信号，以及由此带来的控制信号，如图11.4(c)所示。这就使最终的总线(EISA总线)要比假如从开始时就采用32位地址信号的总线凌乱得多。

并不仅仅只有地址信号线才随着时间的推移不断增加，总线中的数据信号线也有所增加，但原因不一样。一般来说，有两种办法可以提高总线中的数据带宽：缩短总线周期(单位时间内传送次数增加)和增加总线中数据信号线(每次传送更多的数据位)。虽然后一种方法有可能提高总线速度，但由于总线中不同的信号线的传输速度有细微的差别，也就是所

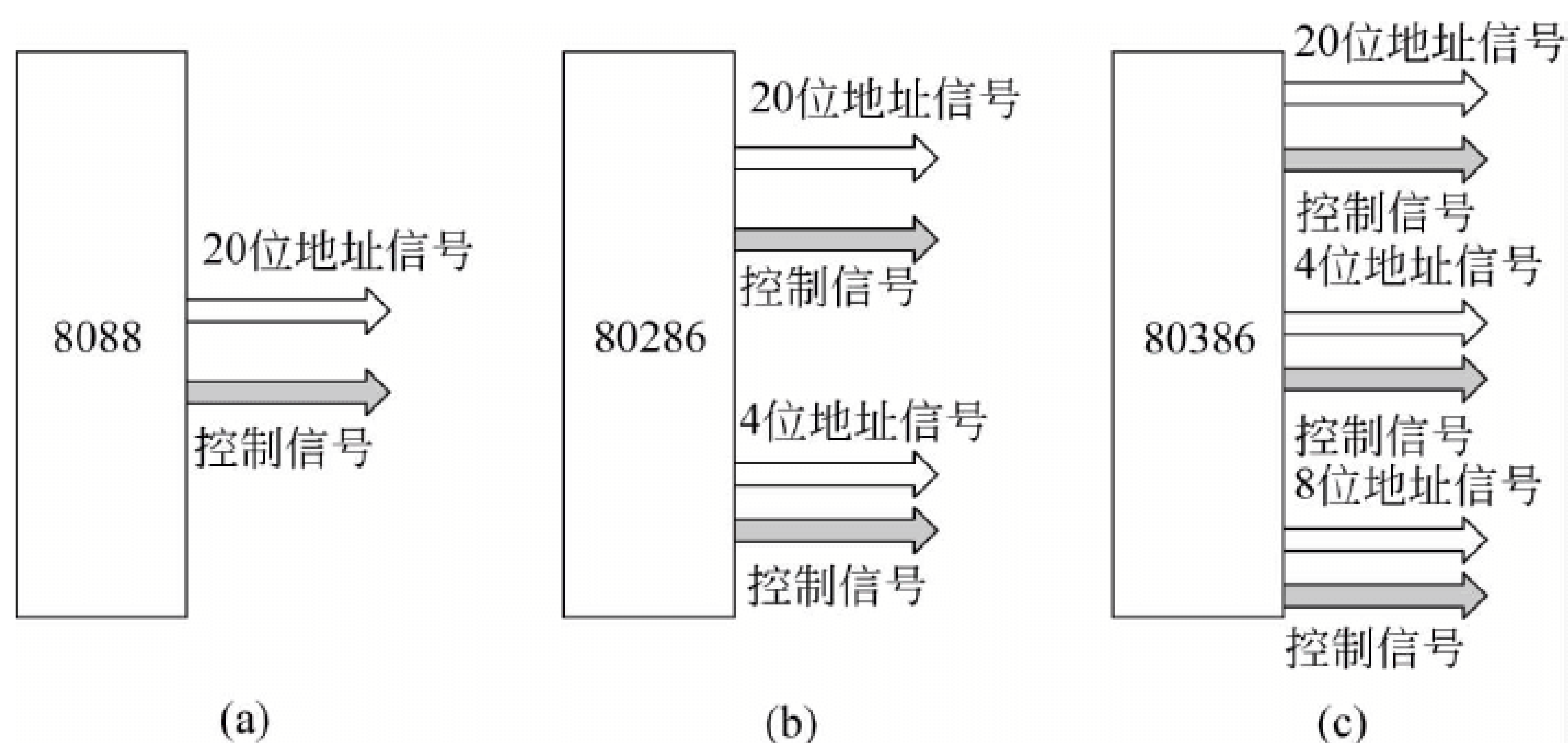


图 11.4　PC 总线随时间的发展变化

谓总线偏离问题的存在,使得这种办法比较困难。总线速度越快,偏离就越严重。

提高总线速度带来的另一个问题是无法保证向后兼容,为慢速总线设计的旧主板上无法使用新的总线。淘汰旧的主板又会使他们的用户和生产厂商不满意。这样,为提高总线数据速度而通常采用的办法就是增加数据信号线了,与图 11.4 中地址信号的增加类似。然而,这种方式最终无法产生一个清晰的设计方案,只会越来越复杂。例如,IBM PC 系列就在几乎同样的总线上弄出了 8 位、16 位和 32 位数据信号的不同版本的总线。

为彻底摆脱超宽总线带来的问题,有时设计者选择采用混合总线方案。这种方案放弃了原来数据信号和地址信号分开的思路,而只是笼统地说有 32 根地址信号和数据信号线。在总线操作开始时,这些信号用作地址信号,然后,它们又可以用作数据信号。例如,对内存写操作,这就意味着地址信号要先传给内存并锁存起来,然后才能在总线上传送数据信号。而用原来地址信号和数据信号相分离的方案时,地址和数据可以同时传送。混合总线减少了总线宽度(和成本),但也降低了系统的速度。设计者必须仔细权衡这两方面的得失,再做出选择。

11.2.4　总线时钟

总线可以根据其时钟类型分为同步总线和异步总线两大类。同步总线中有一条由晶振驱动的产生固定频率的方波的信号线,总线的所有操作都将占用其中的几个完整方波,我们把一个方波的时间称为总线周期。异步总线中不存在一个起控制作用的时钟,它的总线周期可以是总线操作所需的任意长度,并不要求其上面的所有设备都保持一致。下面我们分别对它们进行讨论。

1. 同步总线

我们以 CPU 访问主存储器的过程来说明同步总线的工作原理。图 11.5 是 CPU 访问主存储器的时序,其中,Φ 是用于 CPU 和主存储器同步的方波,其周期为 T。读主存的操作从方波的第一个周期 T_1 的上升沿开始,首先是往地址总线上送要访问的存储单元的地址。经过 T_{AD} 延迟后,地址总线上地址保持稳定;然后,CPU 发出 $\overline{MREQ}$ 和 $\overline{RD}$ 信号。前一个信号说明 CPU 要访问的是内存(反之则是访问外设),后一个信号表示要进行读操作(反之则是写操作)。假定 $T_{AD}<T/2$,我们可以在 T_1 的下降沿开始发出 $\overline{MREQ}$ 和 $\overline{RD}$ 信号。如果主存储器响应速度足够快,能够在下一个总线周期 T_2 送出数据,则 CPU 可在 T_2 的下降沿发

选通信号，从数据总线上得到读出的数据。但一般情况下，CPU 速度比主存储器要高很多，因此，总线周期也主要依赖 CPU 的速度来确定，主存储器可能在 T_2无法给出数据。为通知 CPU 不要期待马上得到数据，主存在 T_2的起始处发出一个$\overline{\text{WAIT}}$信号，这将插入一个等待状态（额外的一个总线周期），直到主存完成数据输出并将$\overline{\text{WAIT}}$信号置反。由于内存较慢，插入了一个等待状态（T_2）。在 T_3的起始位置，主存确知它能在本周期内给出数据后，将$\overline{\text{WAIT}}$信号置反。

在 T_3的前半部分，主存将读出的数据放到数据信号线上，然后，在 T_3的下降沿，CPU 选通（也就是读）数据信号线，将读出的数据锁存（也就是存放）到内部的一个寄存器中。读完数据后，CPU 再将$\overline{\text{MREQ}}$和$\overline{\text{RD}}$信号置反。需要的话，CPU 可以在下时钟的下一个上升沿启动另外一个访问主存的周期。

需要指出的是，图 11.5 是一个实际时序关系的高度简化版本。实际上，还需要有很多其他的时间限定条件。

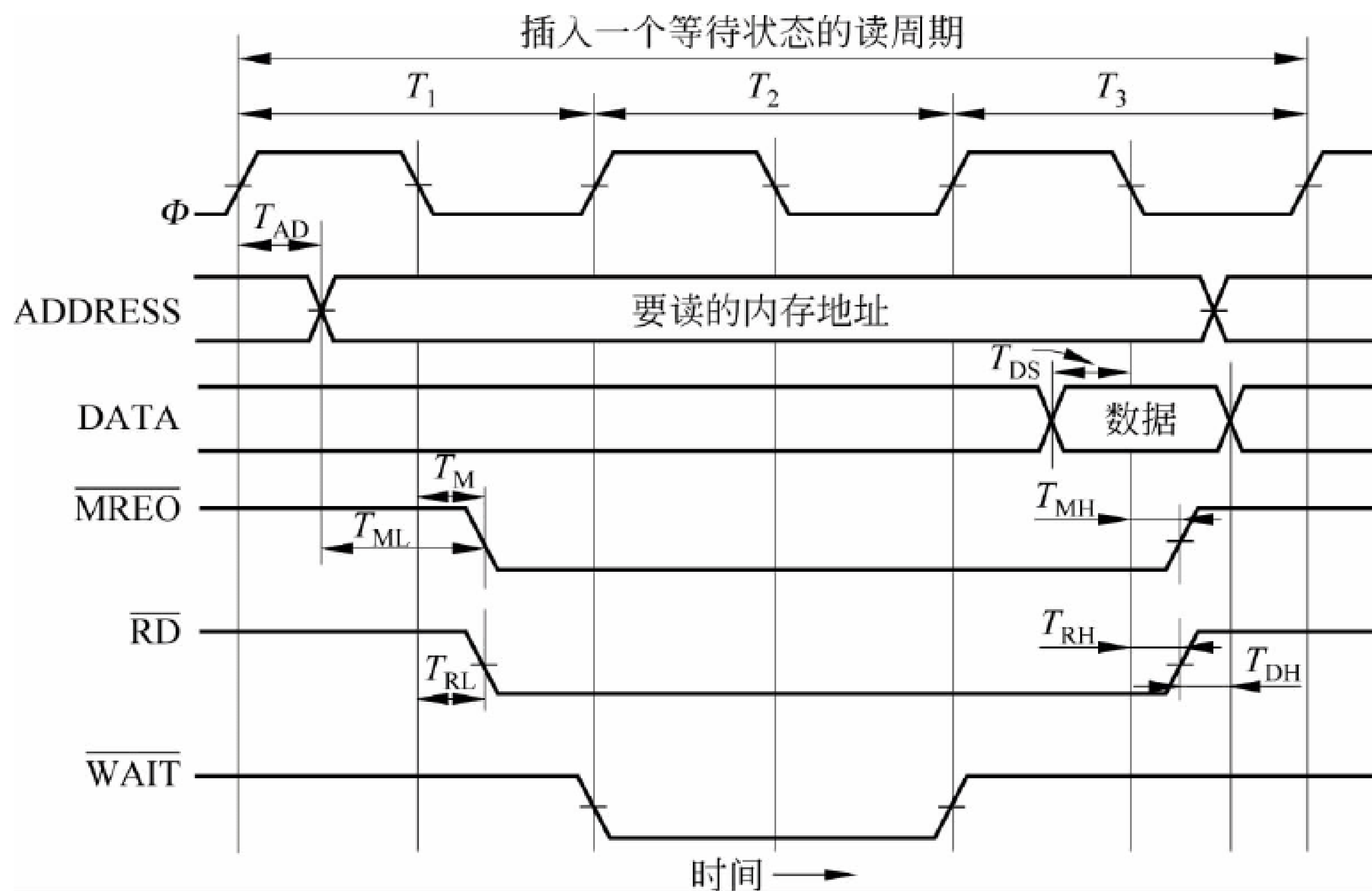


图 11.5　同步总线上的读时序

2. 异步总线

同步总线使用同一个时钟信号，工作原理简单，但也存在一些问题。首先是它要求所有的任务必须在一个或多个总线周期内完成。即使 CPU 和内存芯片可以在 3.1 个总线周期内完成数据读写的话，使用同步总线也必须将其拉长到 4 个总线周期，因为同步总线不允许有不完整的总线周期。另外，如果一条同步总线上接有多个不同的设备，这些设备的数据传输速度有快有慢，那么，总线周期就必须设计得能满足最慢的设备，而快速设备就不可能满效率地运行。

采用如图 11.6 所示的没有主时钟的异步总线后，就可以解决上述问题了。它放弃了同步总线将一切事情都绑定在时钟信号上的做法，而是在总线的主设备给出地址信号、$\overline{\text{MREQ}}$、$\overline{\text{RD}}$及其他所有需要的信号后，再给出一个我们称为主同步信号的$\overline{\text{MSYN}}$（Master SYNchronization）信号。当从设备得到这个信号后，就以它本身最快的速度响应和运行，完成任务后，发出从同步信号$\overline{\text{SSYN}}$（Slave SYNchronization）。

主设备一得到从同步信号$\overline{\text{SSYN}}$,就知道数据已经准备好,它就可以对数据进行锁存,并撤销地址信号,同时将$\overline{\text{MREQ}}$、$\overline{\text{RD}}$和$\overline{\text{MSYN}}$信号置反。从设备得到已被置反的$\overline{\text{MSYN}}$信号后,知道一个访问周期已经完成,就可以将$\overline{\text{SSYN}}$信号置反,这样,就回到了起始状态,所有的信号都处于置反状态,等待主设备启动下一个总线访问周期。

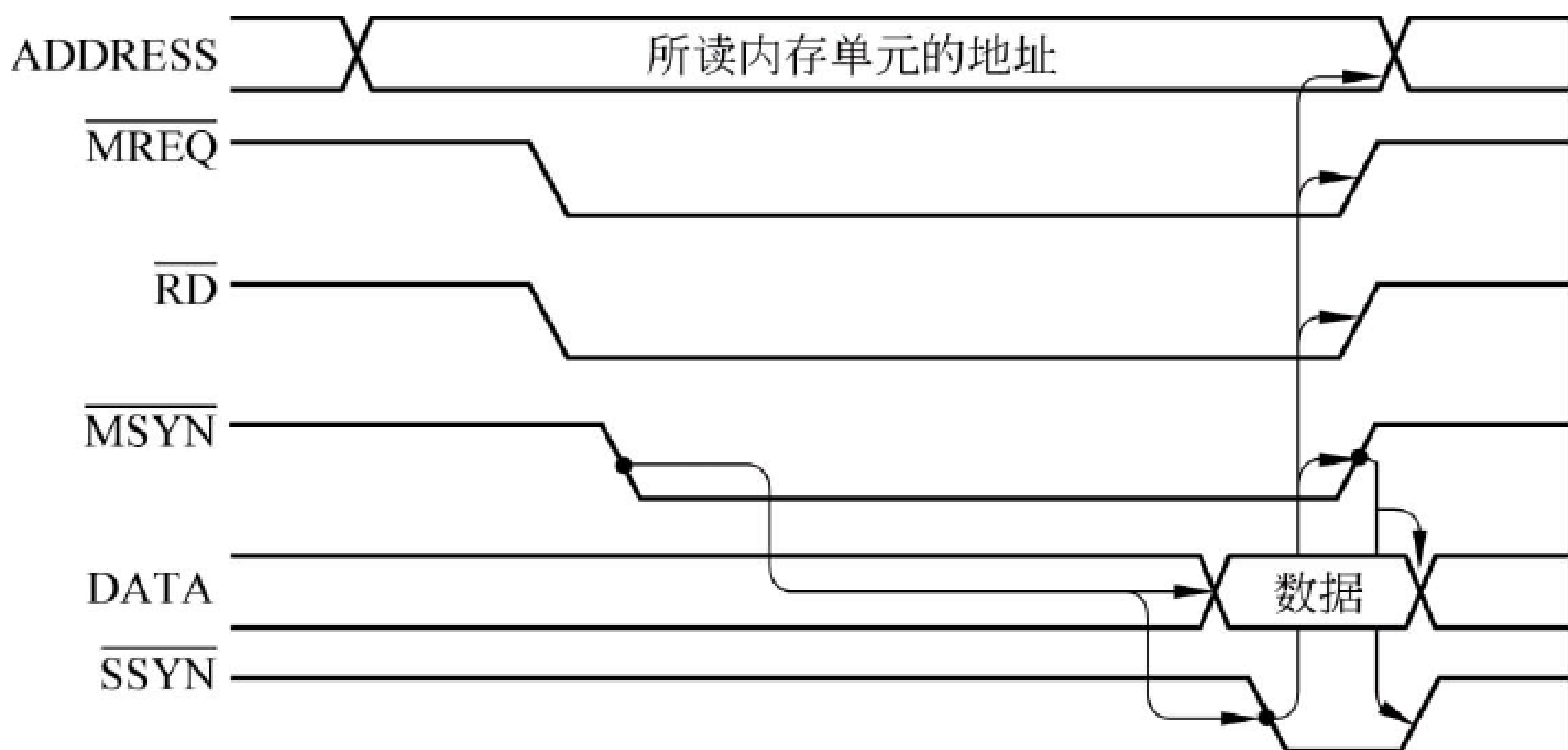


图 11.6　异步总线工作时序

异步总线的时序图用箭头来表示原因和结果,就如图 11.6 一样。$\overline{\text{MSYN}}$信号的给出使数据信号建立,并使从设备发出$\overline{\text{SSYN}}$信号。依次地,$\overline{\text{SSYN}}$信号的发出将导致地址信号的撤销和$\overline{\text{MREQ}}$、$\overline{\text{RD}}$及$\overline{\text{MSYN}}$信号的置反。最后,$\overline{\text{MSYN}}$信号的置反导致$\overline{\text{SSYN}}$信号的置反,结束整个读过程。

我们把一连串以这种方式工作的信号称为**握手**工作方式。它由以下 4 个必需的事件组成。

(1) 发出$\overline{\text{MSYN}}$信号;

(2) 响应$\overline{\text{MSYN}}$信号而发出$\overline{\text{SSYN}}$信号。

(3) 响应$\overline{\text{SSYN}}$信号而将$\overline{\text{MSYN}}$信号置反。

(4) 响应$\overline{\text{MSYN}}$信号置反而将$\overline{\text{SSYN}}$信号置反。

很明显,握手方式和时序无关。每个事件都由前一个事件引起,而不是由时钟脉冲控制。如果有一对主从设备速度较慢,它也不会影响下一对快速的主从设备。

现在,异步总线的优势应该是清楚了,但实际上绝大多数总线都是同步总线,主要原因是同步的系统容易设计和制造。主设备只需负责发出信号,而从设备只需要响应信号。在不存在信号反馈的情况下,如果选择得当,所有设备在没有握手信号的情况下也能正常工作。而且,在同步总线技术上已经有了大量的投资。

11.2.5　总线仲裁

总线仲裁要解决的是连接在总线上的多个主设备同时使用总线的竞争问题。如 CPU 和输入输出系统同时访问主存储器,多处理器系统中两个处理器同时申请使用总线等。为防止发生冲突,就要在总线上引入仲裁机制。仲裁机制可根据总线仲裁器设置的不同位置分为集中式仲裁和分散式仲裁。总线仲裁逻辑集中在一起的仲裁方式称为集中式仲裁,而分散式仲裁的总线仲裁器分散在与总线连接的各个部件或设备中。集中式仲裁是现代计算机系统常常采用的方式,本节主要介绍集中式仲裁方式。常见的集中式仲裁方式有 3 种:

链式查询、计数器定时查询和独立请求。

1. 链式查询方式

链式查询方式如图 11.7 所示。在这种仲裁方式中，有 3 根总线控制线：总线状态线(BS)、总线请求线(BR)和总线同意线(BG)。

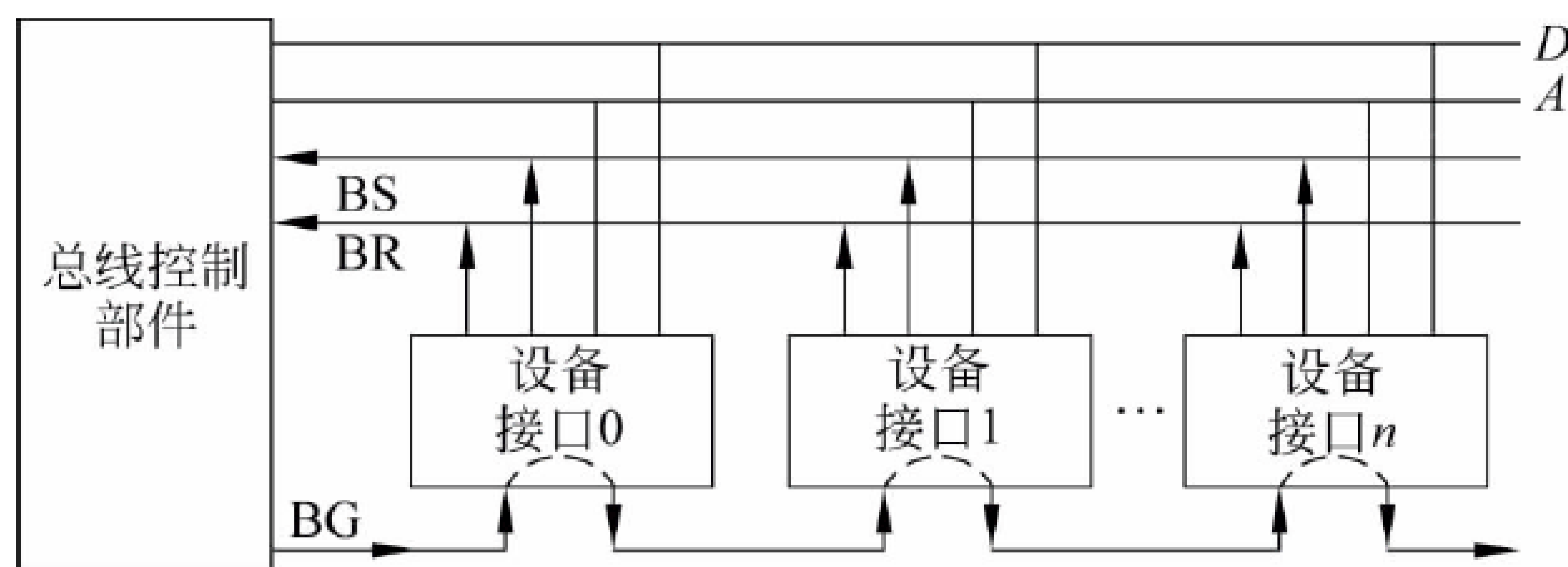


图 11.7　链式查询方式示意图

(1) BS。总线忙/闲状态线，当其有效时，表示总线正被某外设使用。

(2) BR。总线请求线，当其有效时，表示至少有一个外部设备要求使用总线。

(3) BG。总线同意，当其有效时，表示总线控制部件响应总线请求(BR)。

在这种方式中，总线同意信号(BG)串行地从一个 I/O 接口送到下一个 I/O 接口，如果 BG 到达的接口无总线请求，则继续往下传；如果 BG 到达的接口有总线请求，BG 信号便不再往下传，这意味着该 I/O 接口获得了总线控制权。BG 信号就像一条链一样串连所有的设备接口，故这种总线仲裁方式称为链式查询方式。在查询链中，离总线仲裁器越近的设备就具有越高的优先权。

链式查询方式的优点是只用很少几根线就能按一定的优先次序实现总线控制，并且这种链式结构很容易扩充设备。其缺点是对询问链的电路故障很敏感，如果第 n 个设备接口中有关电路出现故障，那么，第 n 个设备以后的设备都不能进行工作；查询链的优先级是固定的，如果优先级高的设备出现频繁的请求，优先级较低的设备就可能长期不能使用总线。

2. 计数器定时查询方式

计数器定时查询方式如图 11.8 所示。在这种仲裁方式中，总线上任一设备要求使用总线时，都可以通过“总线请求线”(BR)发出总线请求信号，总线仲裁器接到请求信号后，在“总线忙”(BS)为 0，即总线空闲的情况下，让计数器开始计数，计数值通过一组地址线发至各设备，当地址线上的计数值与请求总线的某一设备地址一致时，该设备把“总线忙”(BS)置位，获得了总线控制权。此时，终止计数查询。

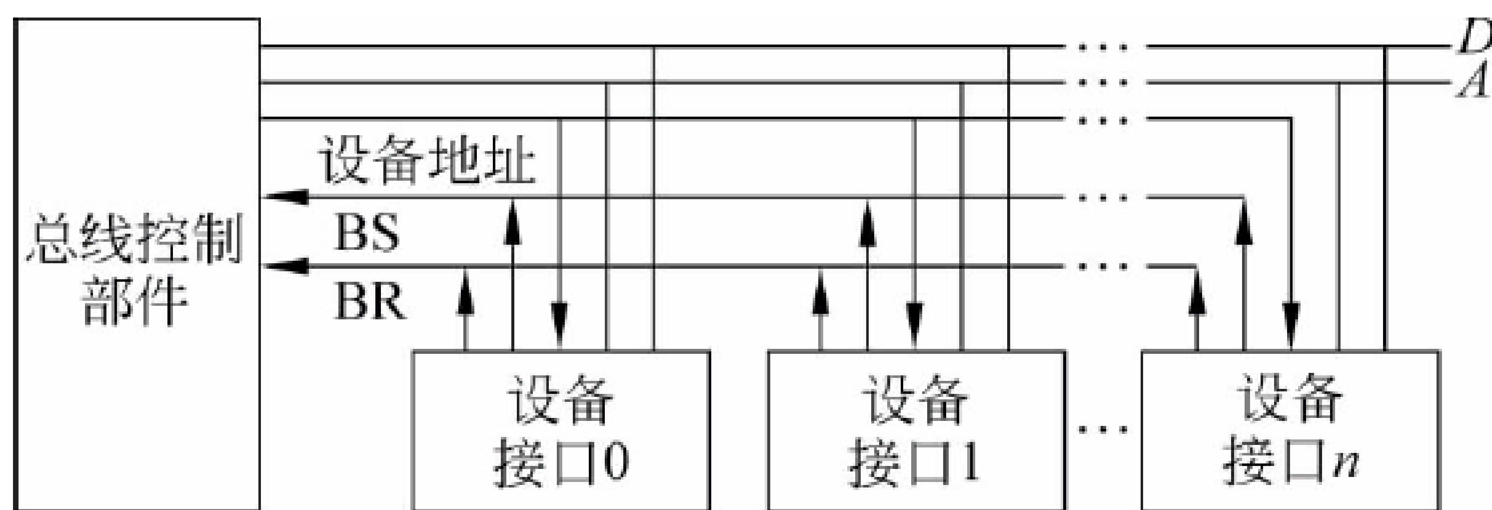


图 11.8　计数器定时查询方式示意图

这种方式的特点：每次计数可以从 0 开始，此时各设备的优先次序与链式查询法相同，即优先次序是固定的；计数也可以从终止点开始，此时各设备使用总线的优先级是相同的。

计数器定时查询方式的优点是：优先次序可以方便地改变；这种查询方式对电路故障不如链式查询方式敏感。其缺点是：要增加一组设备地址线，从而增加了控制线的数量，而且控制也较为复杂。

3. 独立请求方式

独立请求方式如图11.9所示。在这种仲裁方式中，每一个共享总线的设备均有一对“总线请求线”(BR)和“总线同意线”(BG)。当设备要求使用总线时，便发出“总线请求”信号，总线仲裁部件中一般有一个排队电路，可以根据一定的优先次序决定首先响应哪个设备的请求。

独立请求方式的优点是：响应时间快，对优先次序的控制也相当灵活，它可以预先固定优先次序，也可以通过程序来改变优先次序。其缺点是：控制线的数量多，比如要控制 n 个设备，必须有 n 根BR线和 n 根BG线，相比之下，链式查询方式只需2根，计数器定时查询方式只需约 $\log_2 n$ 根；独立请求方式仲裁器也要复杂得多。

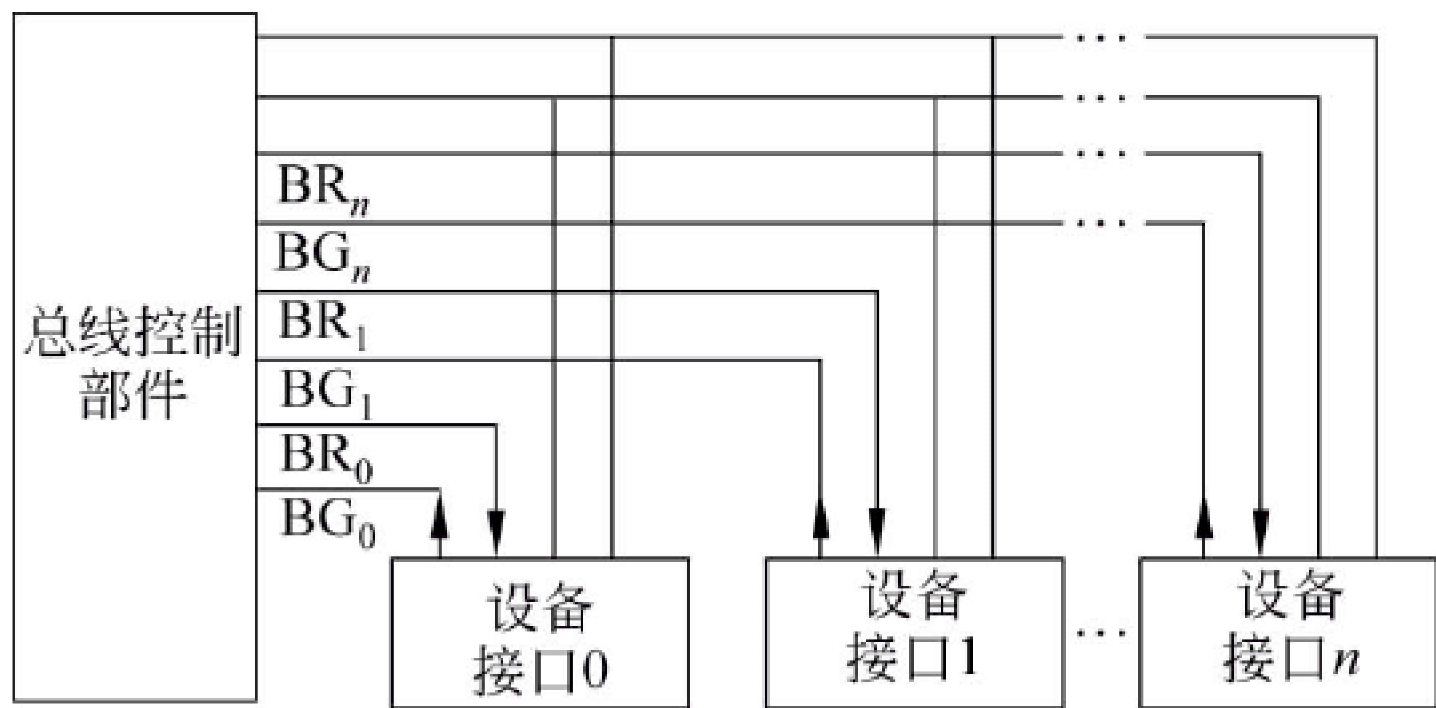


图11.9　独立请求方式示意图

11.2.6　总线举例

总线是连接计算机各功能部件的信息通道。计算机早期只有一条系统总线，发展到目前的多总线结构，诞生过许多著名的总线。本节我们介绍几种PC中还在使用的总线。

1. ISA总线

前面介绍过，早期的PC使用的是IBM PC总线，是IBM公司为其PC制定的，后成为PC兼容机总线的事实标准。它最初为8位数据线(后扩充到16位)、20位地址线(后扩充到24位)，工作频率为8.33MHz。

后来，随着PC市场的日益扩大，IBM希望能占有更大的市场份额，在其新推出的PS/2系列上，采用了一种全新的总线Microchannel，并不再公开新总线的技术标准，以此来保护自己的利益。

但是，其他PC厂商却乘机开发出新的总线标准，定名为工业标准结构(Industry Standard Architecture，ISA)总线，工作频率还是8.33MHz，使原有的许多设备还能继续得到使用。其后，以ISA为基础推出了EISA总线，地址线扩展到32位。

ISA和EISA工作频率低，数据传输率也不高，对于早期PC基于文本的应用还足以应付，但随着Windows系统的推出，图形界面成为PC的基本应用界面，对总线的传输率提出了更高的要求，ISA和EISA总线逐渐无法满足应用的要求。

2. PCI 总线

1990 年推出了一种新的总线，其带宽比 EISA 总线要高，这就是 PCI 总线（外围部件互连总线，Peripheral Component Interconnect Bus）。Intel 还成立了一个行业联盟，即 PCI 特别利益集团，来管理 PCI 总线的未来，这也使得 PCI 总线的使用越来越普遍。事实上，从 Pentium 开始，每台基于 Intel 的计算机，以及其他的一些计算机，都装配了 PCI 总线。

最早的 PCI 总线每个周期传输 32 位，速度为 33MHz（总线周期为 30ns），总的带宽为 133MB/s。1993 年推出了 PCI 2.0，到 1995 年又推出了 PCI 2.1。PCI 2.2 中有些专为便携式计算机设计的特性（主要是节省电池的能量）。PCI 总线的最高速度是 66MHz，并能以 64 位传输，若磁盘和系统的其他设备可以保证速度，全屏、全动画显示是完全可行的。在任何情况下，PCI 总线都不会成为瓶颈。

尽管 528MB/s 的速度看起来已经十分快了，但它还是存在两个问题。第一，它不适合做内存总线。第二，它无法兼容所有旧的 ISA 卡。Intel 提出的解决方案是为计算机设计 3 条或更多的总线，如图 11.10 所示。图中，CPU 可以通过特设的内存总线和内存交换数据，而且 PCI 总线上还可以连接一种 ISA 总线。这个设计可以满足上面所有的要求，因此，所有早期的 Pentium 计算机都采用了这种体系结构。

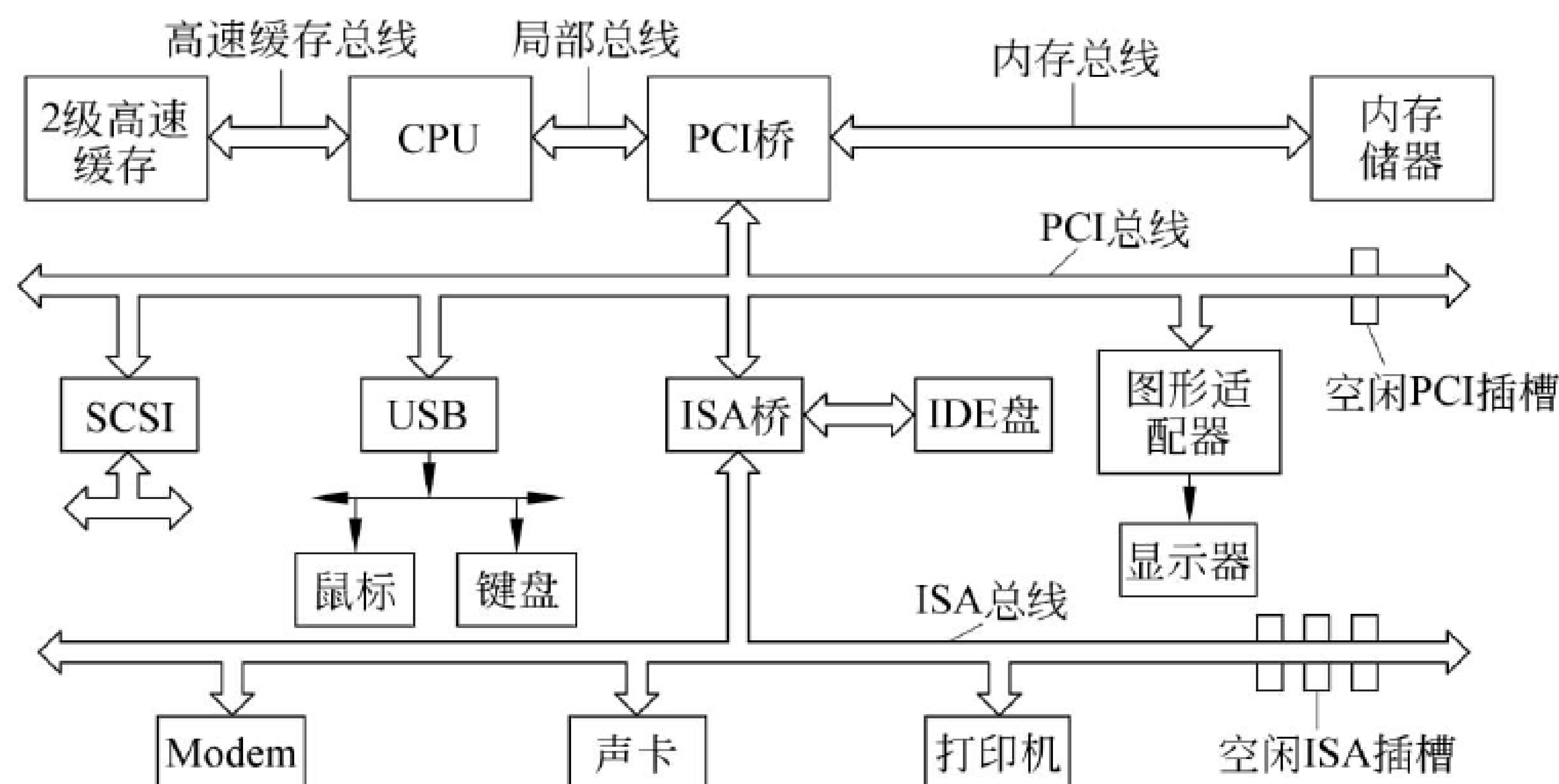


图 11.10　早期 Pentium 的总线结构

这个体系结构的两个关键部件是两片搭桥芯片，PCI 桥连接了 CPU、内存和 PCI 总线；ISA 桥则将 PCI 总线和 ISA 总线连接在一起，而且还能支持一到两个 IDE 盘。采用该体系结构后，CPU 和主存之间有了专用的高速通道，提高了 CPU 访问存储器的带宽。而通过 PCI 桥，PCI 总线与 CPU 和主存连接，可以为高速的外部设备（如 SCSI 盘等）提高较高的带宽。另外，ISA 桥实现了 PCI 总线和 ISA 总线的互连，为使用以前的 ISA 设备提供了接口。

到 20 世纪 90 年代后期，几乎所有的人都认为，ISA 总线时代已经过去，因此，新的设计中把它排除在外。然而，也就是在那个时候，显示器的分辨率提高到了 1600×1200 左右，对全屏幕显示全动画图形的要求也就更高，尤其是在一些高度交互的游戏中。因此，Intel 又设计了另外一种总线，专门用来驱动图形卡。这就是 **AGP 总线**（加速图形端口总线）。第一个版本的 AGP 总线，也就是 AGP 1.0，带宽为 264MB/s，被定义为 1×。虽然比 PCI 总线速度要慢一些，但它是供图形卡专用的。经过这些年的发展，新的版本不断出现，现在的 AGP 3.0 的速度已达到 2.1GB/s（8×）。图 11.11 给出了 Pentium 4 系统的总线结构示

意图。

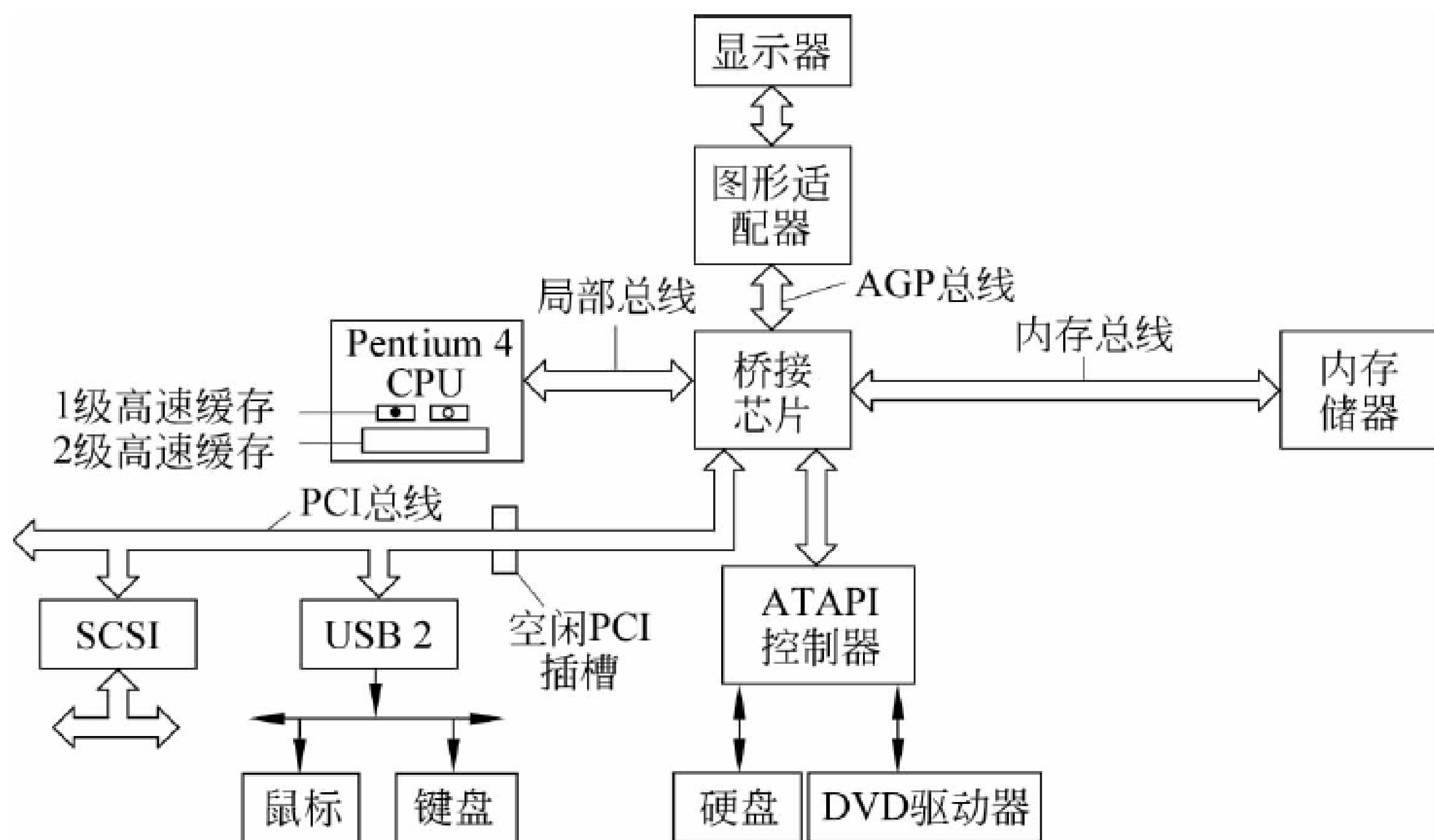


图 11.11　Pentium 4 的总线结构

在这种设计框架下,桥接芯片成为结构的中心。它连接了系统的5个主要部分:CPU、内存、图形适配器、ATAPI控制器及PCI总线。在某些情况下,还可让它支持以太网及其他高速的设备。低速的设备被连接到PCI总线。

从内部看,桥接芯片可以分为两部分:内存桥和I/O桥。内存桥连接CPU到内存及图形适配器的总线,I/O桥连接ATAPI控制器、PCI总线及其他(可选)快速I/O设备,为这些设备之间提供了快速的直接连通通道。两桥之间以非常高的速度实现互连。

PCI总线是同步总线,它上面的所有事务发生在一个主设备,正式名称为发起者,和一个从设备,正式名称为接收者,两个设备之间。为使PCI总线的信号线数目减下来,地址信号和数据信号使用共用信号线。通过这种方式,PCI卡上只需要64位就可以满足地址信号和数据信号的需要,虽然PCI支持64位地址和64位数据。

地址信号和数据信号混合后工作流程如下。对读操作,在第一个总线周期,主设备将地址送上总线传递给从设备;第二个总线周期,主设备撤销地址信号并将控制权交给从设备;第三个总线周期,从设备将读出的数据送上总线传递给主设备。对写操作,主设备不需要交出总线的控制权,因为地址和数据都是由主设备发出的。不过,最小的事务依然需要3个总线周期。若从设备没有在3个周期内响应请求,它就要插入等待状态。不限大小的块传输和其他的几种总线类型也是可以的。

PCI总线采用集中仲裁方式,总线控制器一般设置在PCI桥上。每个PCI设备都有两根线连接到仲裁器。其中,REQ#用于发出总线请求,而GNT#用来接收总线授权。申请总线时,先由PCI设备(包括CPU)发出REQ#信号,并等待总线仲裁器对它发出GNT#信号。只有得到授权后,设备才能在下一个周期内使用总线。

PCI总线的信号线包括必备信号和可选信号两大部分。其中,必备信号可分为以下5组。

(1) 系统信号线。CLK和RST#。

(2) 地址与数据线。包括32根地址和数据的复用线以及其他用来解释数据与地址的信号线。

（3）接口控制线。控制工作时序，并在主从设备间进行协调。

（4）仲裁线。与PCI其他信号线不同，这组线不是共享的，每台接入到PCI总线的设备各有一对仲裁线直接连接到仲裁器上。

（5）错误报告线。报告数据或地址校验错误。

可选信号线中，包括32根用于将地址和数据复用线扩充到64位的信号线和其他辅助信号线。

3. PCI Express

PCI总线遇到的另一个问题是它的接口卡太大了，它们无法应用到膝上计算机，更不用说是掌上计算机了，它的制造商喜欢的是更小的设备。而且，有些制造商还想重新组装PC，将CPU和内存封闭到一个微型的盒子中，而将硬盘放置到显示器中去。如果还采用PCI总线，这也是不可能实现的。

解决上述问题的方案之一就是 **PCI Express**。它和PCI总线几乎没有关系，事实上，它也不应该算是总线，但市场营销人员不愿意轻易放弃已经很知名的PCI这个名称。已经有许多PC中配置了PCI Express，下面，我们来讨论它的运行原理。

PCI Express方案的核心是彻底摒弃了在并行导线上连接众多主设备和从设备的传统的总线模式，代之以基于高速点到点串行连接设备的方式。它对传统的ISA/EISA/PCI总线进行了根本的改变，而从局域网，尤其是交换以太网借鉴了许多重要的思想。其最基本的出发点是：从最深层次看，PC是CPU、主存储器及输入输出接口芯片互相连接而构成的，PCI Express要做的仅仅是为这些芯片的串联提供一种通用的交换机制。图11.12给出了一个典型的PCI Express的结构。

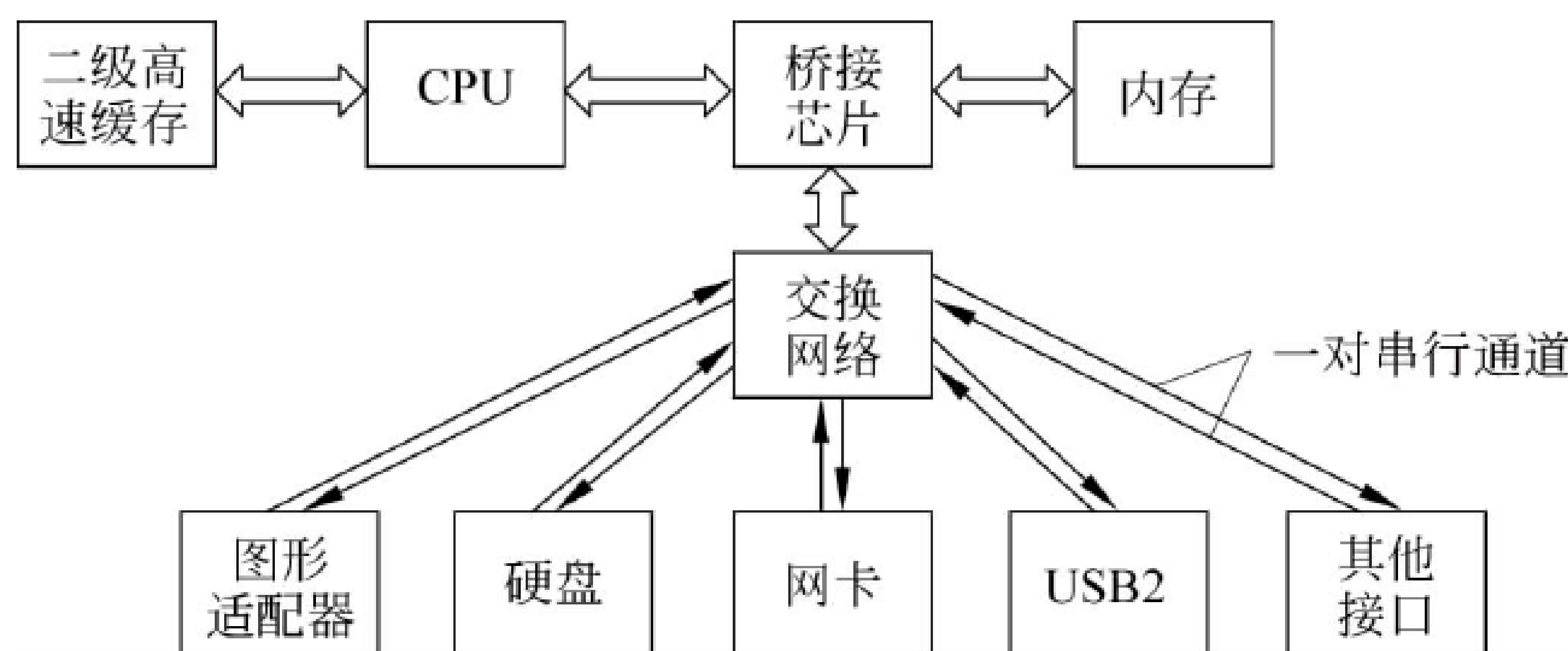


图11.12　典型的PCI Express系统

在图11.12中，CPU、内存和高速缓存依然通过传统的方式连接，不同的是用交换网络和桥接芯片（也可能交换网络就是桥接芯片的一部分）连接在一起，而每个输入输出接口芯片有专门的点到点连线连接到交换网络。每个这种连接由一对单向的通道组成，一个通道从交换网络到设备，另一个从设备到交换网络。每个通道由两根导线组成，一个传送信号，另外一根是地，以在高速传输中抵抗干扰。这种结构下，所有设备被同等对待，形成了一个更为统一的模式，应该会替代原有的总线模式。

PCI Express体系结构和原有的PCI总线体系结构有3点关键的区别。我们已经指出了其中的两点：集中的交换网络取代了“多站”的总线，串行的点到点连接取代了并行的总线。第三点关键区别比较隐蔽。PCI总线传输从概念上讲，是由主设备发出一个命令给从设备，读一个字或多个字。而PCI Express的模型是一个设备发送一个数据包到另一个设

备。**包**由包头和有效载荷组成,是网络中常用的概念。**包头**包含有一些控制信息,这样,PCI 总线上的许多控制信号就可以省略了。**有效载荷**由被传送的数据组成。从本质上看,PCI Express 结构的 PC 就是一个微型的包交换网络。

除这 3 点主要区别外,它们之间还有另外一些不同之处。①包中还有一些检错纠错码,可使 PCI Express 的可靠性比 PCI 总线要好。②接口卡和交换网络的连线距离比较长,最长可达到 50cm,这就允许系统分离。③由于设备本身可以是一个新的交换网络,系统的可扩展性比较强。④设备可热插拔,意味着它们可在系统运行时接入或断开。⑤串行的接头比原来的 PCI 接头要小得多,设备和计算机都可以做得小一些。

4. USB 总线

PCI 总线为外部设备提供了高速的带宽,但对于低速设备,如键盘、鼠标、数码照相机等,使用 PCI 总线的成本显得太高。另外,PCI 总线和 ISA 总线都要求新设备使用接口卡接入到总线上,这意味着用户要负责设置卡上的开关和跳线来保证新的接口卡不和原有设备接口卡发生冲突,增加了用户使用计算机的困难。

为解决这个问题,在 20 世纪 90 年代中期,7 家著名的计算机公司(Compaq、DEC、IBM、Intel、Microsoft、NEC 和 Northern Telecom)共同提出了一个将低速输入输出设备连接到计算机的方案。这以后,数百家其他公司加入了这个阵营。这个方案就是通用串行总线(Universal Serial Bus,USB),目前已在个人计算机上得到广泛的实现。

最早参与设计并将 USB 付诸实现的公司对 USB 总线提出了如下技术指标。

(1) 用户不必再设置卡上、设备上的开关或跳线。

(2) 用户不必再打开机箱来安装新的输入输出设备。

(3) 应该只需要一根电缆线就可以将所有设备连接起来。

(4) 输入输出设备应可以从电缆上得到电源。

(5) 单台计算机最多可连接 127 个设备。

(6) 系统应能支持实时设备(如声卡、电话)。

(7) 计算机运行的时候也可以安装设备。

(8) 安装新设备后不必重新启动计算机。

(9) 新的总线和连接它的输入输出设备的生产成本不应该太高。

这些指标显然是为适应低速的设备,如键盘、鼠标、照相机、快照扫描仪、数字电话等提出的。最终设计的 USB 总线的总的带宽是 1.5MB/s,对于一定数量的这类设备是足够了。选择如此低的带宽也是为了降低成本。

1998 年,USB 标准完成后,USB 标准的设计者又开始了一个更高速度的 USB 版本的设计,即 **USB 2.0**。这个标准和已有的 USB 1.1 版本类似,且完全兼容它,只是在前两个版本提供的两种传输速度之外,又增加了 480Mb/s 的第三级传输速度。

有了 480Mb/s 的 USB,显然就和颇为流行的、名为 FireWire 的 IEEE 1394 串行总线有了竞争关系,它的速度为 400Mb/s。尽管目前每台 Pentium 的计算机都装上了 USB 2.0,但 IEEE 1394 好像也没有消失的迹象,因为它后面有消费电子行业的支持。便携式摄像机、DVD 播放机等设备在可预见的将来还将继续装配 IEEE 1394 接口,因为这些设备的制造商不愿意为很少的改进而承担更换标准的费用,而消费者也不愿意去换标准。

USB 由一个插在主总线(如 PCI)上的根集线器组成(见图 11.10)。这个集线器的电缆

插口可以连接输入输出设备或扩展集线器，以提供更多的插口，这样，通用串行总线系统的拓扑结构就成了一棵以在计算机内部的根集线器为根的树。

USB中有4根导线：其中两根用于数据传输，一根用作电源，还有一根用作地。数据传输编码以电压的转换来表示0，以恒定的电压来表示1，有点像磁记录方式中的见0就翻转的不归0制。

当接入一个新的外部设备时，根集线器检测到这个事件，并发出中断信号给操作系统。然后，操作系统查询设备对其进行识别，并得到这个设备所需要的通用串行总线带宽。若操作系统判断还有足够的带宽来接入这个设备，它将赋给设备一个唯一的地址（1～127之间），并将这个地址和其他一些信息记录到设备内部的配置寄存器中。通过这种方式，可以实现设备的热插拔，而不需要重新启动计算机，更不需要用户进行配置。未初始化的设备的地址是0，系统通过这来判断是否有新设备。

我们用图11.13所示的例子来说明USB总线的协议。每隔1ms，根集线器会广播一个新帧，对所有的设备进行时间同步。帧和位流有关，由数据包组成，第一个数据包总是从根集线器到设备。该帧的后续包可能还是从集线器到设备，也可能是从设备发回到根集线器的。图11.13中即是一个4帧组成的序列。

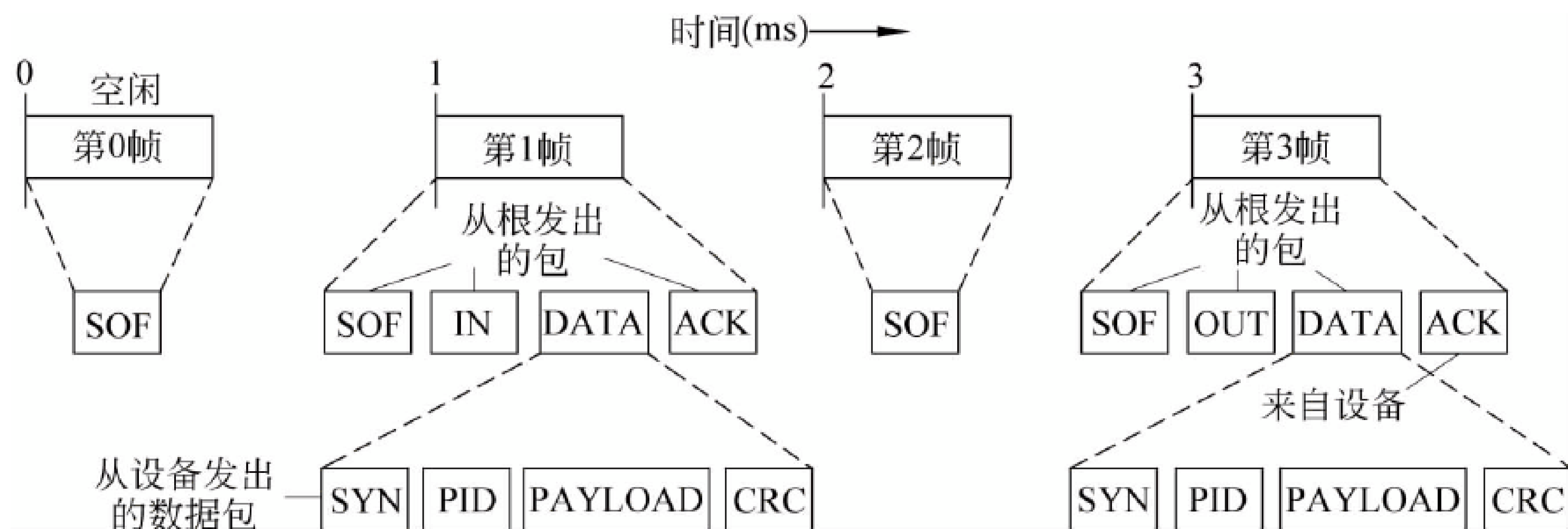


图11.13 USB总线的数据帧

图中第0帧和第2帧中没有工作要做，因此只需要发送一个帧起始（Start Of Frame，SOF）包。这个包总是广播到所有的设备。第1帧是一个询问，例如，请求扫描仪返回它正在扫描的图形的数据。第3帧由发送到某个设备的数据组成，比如发送到打印机的数据。

USB支持4种类型的帧：控制、同步、块传送和中断。控制帧用于配置设备，对设备发出命令，并查询它们的状态。同步帧用于那些需要以精确的时间间隔发送和接收数据的实时设备，比如麦克风、扬声器和电话等。它们之间存在可以准确预测的延迟，但在出错时数据无法重传。块传送帧用于对数据没有实时要求的设备，如打印机等的大批量数据传送。最后，由于通用串行总线并不支持中断，所以还需要中断帧。例如，如果在键盘上的键按下时不由键盘产生中断，也可以让操作系统每50毫秒轮询一次键盘，采集已经键入的键。

帧由一个或多个包组成，有些可能是两个方向传输数据。一共有4种类型的包存在：令牌包、数据包、握手包和特别包。令牌包从根传送到设备，用于系统控制。图11.13中的SOF、IN和OUT包都是令牌包。SOF包是每帧的第一个包，标志着一帧的开始。如果不需要做其他工作，SOF包就是该帧中唯一的包。IN令牌包用于查询，请求设备返回要求的数据。它包含的域中指出它需要查询的是哪个位流，这样，设备就可以知道需要它返回什么

数据(如果设备有多个数据流)。OUT令牌包表示后面跟的是给设备的数据。还有第4种令牌包SETUP(图11.13中没有标出),用来配置设备,对设备初始化。

数据包DATA用来传送最多可达64字节的信息,图11.13给出了数据包的格式,它用8位(SYN)进行同步、8位(PID)说明数据类型、然后是真正要传送的数据(PAYLOAD),最后是16位的循环冗余码(CRC),用于检测数据错误。另外,还定义了3种类型的握手包:ACK(前面的数据包已正确接收)、NAK(检测到CRC错)和STALL(设备忙)。

最后,我们再回到图11.13所示的例子。即使没有什么可干时,每1毫秒都必须有一帧从根集线器发出。第0帧和第2帧中只有SOF包,表示没有工作要做。第1帧是查询帧,所以它以SOF包开始,然后是从计算机到外部设备的IN包,后面是从设备发送到计算机的DATA包,由ACK包通知设备数据已被正确接收。如果出错,将向设备发回一个NAK包,然后,该包将以块传送帧的方式重传。但如果是实时数据,则不进行重传。第3帧在结构上和第1帧类似,只是在这帧中数据是从计算机传送到设备。

USB 3.0版本提供了更高的传输速度,可以达到5Gb/s,实现了全双工数据通信。USB 3.0还能够对设备提供更大电力支持,增加了电源管理功能。同时,USB 3.0支持向下兼容USB 2.0和USB 1.1设备。

11.3 输入输出接口

11.3.1 输入输出接口的功能

提供主机识别(指定、找到)要用的I/O设备的支持,这是通过为每个设备规定几个地址码或编号来实现的。常用的有两种编址方式。

(1) 对计算机的主存储器与I/O设备按统一的格式和方法来分配与安排地址编码。最典型的例子是在PDP-11计算机中的用法,它把计算机可寻址的最高位置的几千字节的存储空间分配给所有的I/O设备,而把其他空间分配给主存储器。这样CPU就可以用统一的MOV指令访问主存储器和I/O设备,而不使用专用的输入输出指令访问I/O设备。在一条MOV指令中,访问的到底是主存储器还是I/O设备,是由用到的地址范围决定的。

(2) 设置并使用专用的输入(IN)输出(OUT)指令访问I/O设备(执行输入输出操作)。由于I/O设备的数量比较少,就可以用比较少的地址为它们编址,该地址被称为I/O端口地址。这种方案被大部分计算机普遍采用。

建立主机和设备之间的控制与相互了解的机制,一方面,主机可以向设备发出操作命令,主机可以了解设备的运行状态;另一方面,设备也可以向主机提出自己的操作要求。这是通过在接口卡中设置命令寄存器、状态寄存器和中断逻辑线路来完成的。

提供主机和设备交换信息过程中的数据缓冲机构,为此,在接口卡中要设置输入数据缓冲寄存器、输出数据缓冲寄存器等部件。

提供主机和设备交换信息过程中的其他特别需求支持,例如,当主机和设备的信号幅度不同时的信号电平转换功能,数据传送中的格式(并行、串行)转换功能,直接内存访问中的额外需求等。

一般来说,这里提到的前3个组成部分是大部分接口卡上都具有的,最后一条则按具体

需要特殊安排。图11.14给出的是一个基本的输入输出接口结构的示意图。

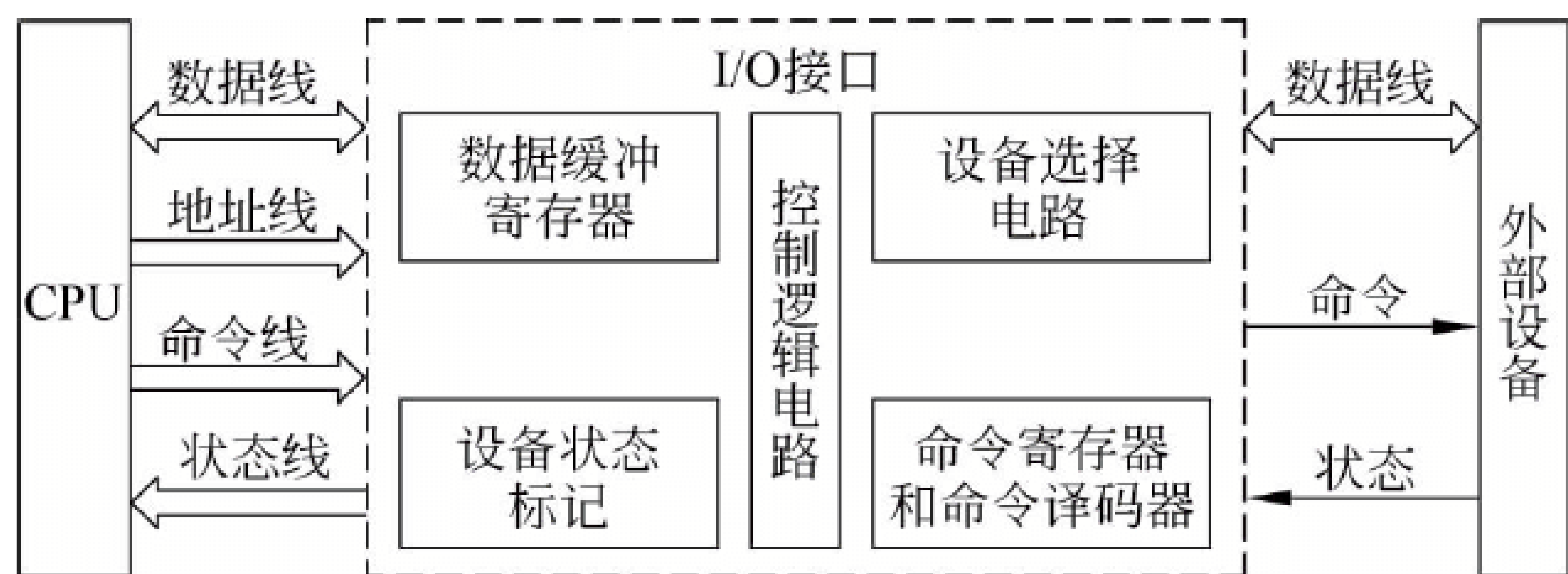


图11.14　输入输出接口结构示意图

11.3.2　通用可编程接口组成

由于外部设备功能各不相同，对接口的要求也不一样，但人们总是希望用同一块接口卡提供更多的功能，以使它能适用于更多种类的外部设备。这需要通过在接口卡上设置一些参数，供用户程序来选择接口功能，这种接口卡就是常说的通用可编程接口。所谓的可编程，就是在程序中可通过指令指定接口的功能、设置接口的运行控制参数等。通过通用可编程接口，计算机主机仅需配置少量的一些接口，就可接入多种外部设备。

通用可编程接口中，对外部设备的识别主要是通过对指令中给出的输入输出端口地址进行译码产生片选信号来完成的。将译码得到的信号分别接到相应接口卡的片选管脚，使该接口在指令给出正确的端口地址时，可正常运行。早期的计算机中，经常采取主机发出广播信息，各接口卡自行比较是否和本接口设置的地址相同来进行判断。

接口卡上通常有接口命令寄存器，存放CPU发来的控制命令；同时，还设置有状态寄存器，由外部设备来设置运行状态，CPU可通过检测该寄存器来了解设备状态。通过命令寄存器和状态寄存器，可使CPU能自如地控制外部设备。

接口卡上还有一个或几个用于输入输出数据缓冲的寄存器，这样，可支持数据在总线上的成组传送，提高总线的利用率和CPU的效率。

最后，接口卡上一般还有处理中断请求、屏蔽和判断优先级的逻辑线路，这是使作为总线从设备的外部设备主动向CPU提出操作请求的重要机制。

11.3.3　输入输出接口举例

当前，各种计算机中使用的外部设备接口数量众多。前面章节中已经介绍的硬盘接口，如IDE、EIDE等在PC中依然使用，SCSI接口在PC服务器上也已经得到广泛使用。而USB接口也逐渐成为PC的标准配置，连接键盘、鼠标、数字相机、移动硬盘等低速设备。当然，使用最多的通用接口还是串行接口和并行接口，它们是几乎每一台计算机的标准接口。

1. 串行接口

串行接口只需要一对信号线来传输数据，主要用于传输速度不高、传输距离较长的场合。目前几乎所有计算机都采用EIA RS-232C作为串行接口标准，它包括了按位串行传输的电气和机械方面的规定。完整的RS-232C接口有25根线，采用一个25芯插头座，但大多数情况下只需使用其中的3～5根即可正常工作，其中最主要的是“发送数据”和“接收数据”线，它们用来在两个系统之间串行传送信息，传送速率在50、75至19200b/s之间，有多

种速率可选择。

对串行接口的使用是通过对其控制寄存器和状态寄存器进行的。先由主机对串行接口的控制寄存器发出指令,设置好串行接口的工作方式,如同步还是异步、数据传送速率、奇偶校验方式以及字符长度等。然后,再发送命令使之开始工作。数据传送期间,主机可根据状态寄存器了解串口的工作状态,并据此确定下一步操作。下面我们以 Intel 公司的 8251 芯片为例来说明串行接口芯片的组成和用法。

Intel 8251 的结构图如图 11.15 所示,其管脚图在图纸的器件图部分给出,为 28 条管脚双列直插封装。从 8251 的结构图上看到它的 5 个组成部分,即接收器、发送器、调制解调控制、读/写控制以及几个入/出缓冲器。这最后一部分又可细分为状态缓冲器、发送数据/命令缓冲器和接收数据缓冲器 3 部分。

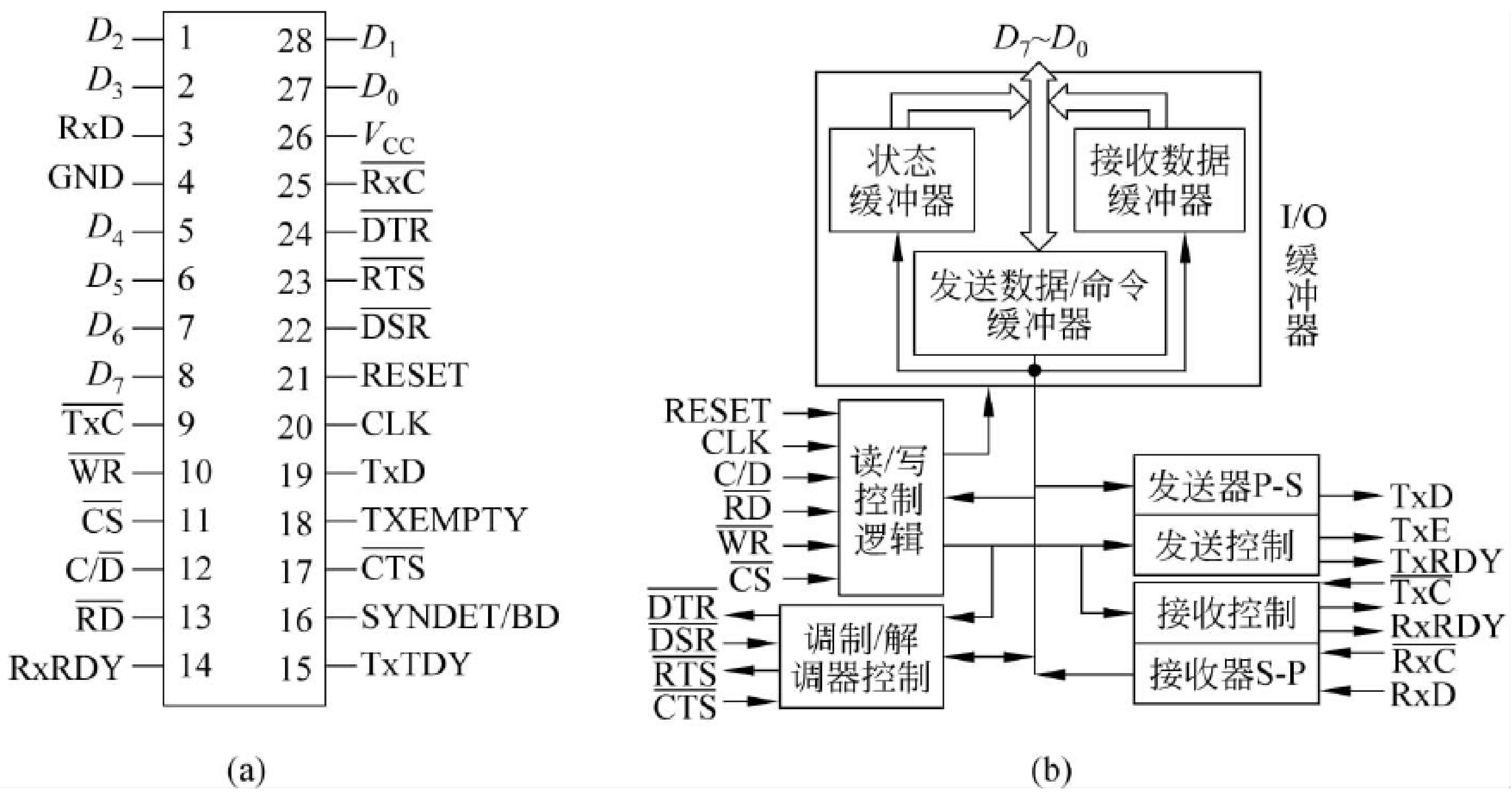


图 11.15 Intel 8251 管脚分布及内部组成框图

从使用角度看,Intel 8251 是一个可编程的多功能通信接口电路。在使用前必须用方式指令和命令指令对其进行初始化操作。

1) 方式指令

方式指令是指把规定的 8 位信息送到 8251 的控制寄存器,以指定其工作方式,例如是同步还是异步传送、所用的波特率、每个字符的长度、对使用奇偶校验的规定、对停止位的规定等。信息的组成规定如图 11.16 所示。

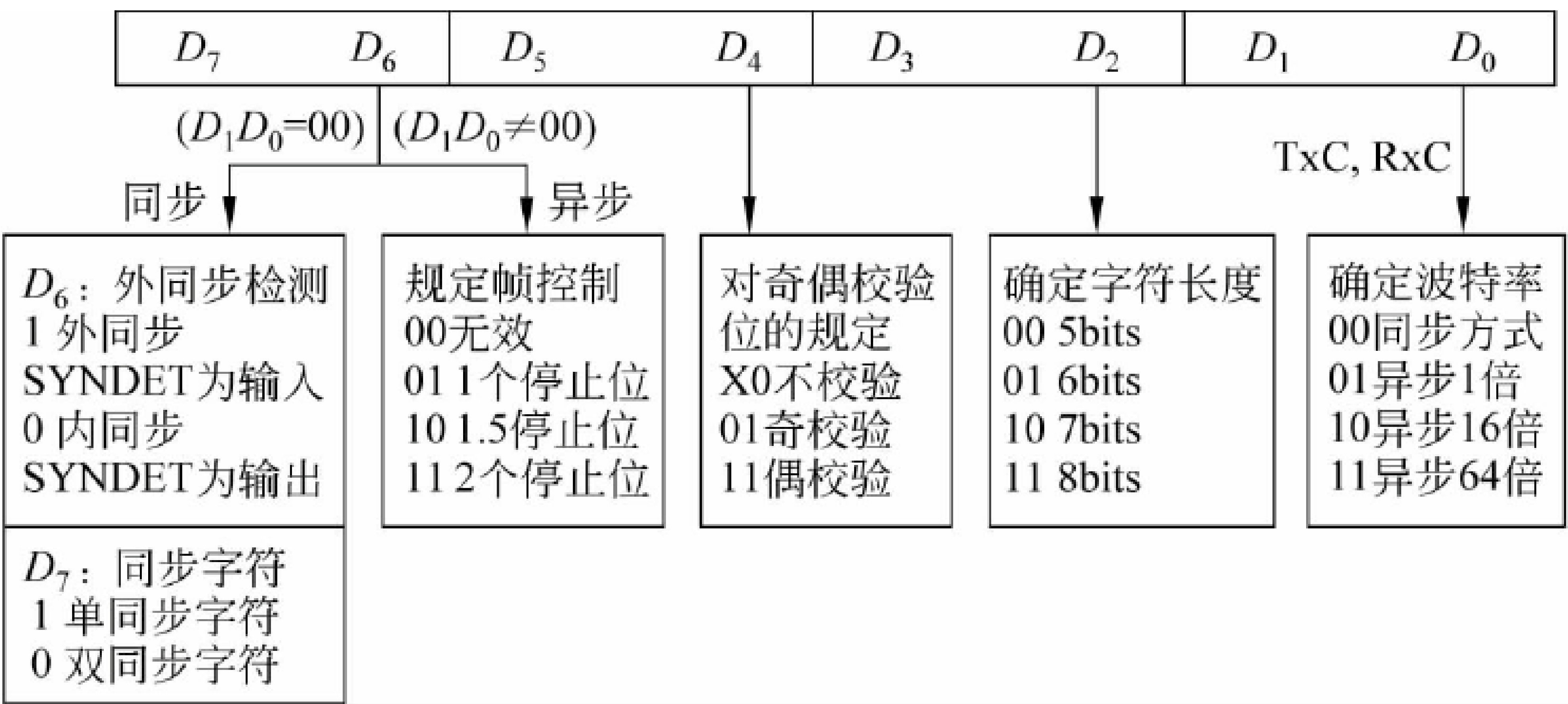


图 11.16 方式指令的格式

2）命令指令

命令指令是把规定的8位信息送入8251接口的控制寄存器，以使其进入运行状态，能执行发送或接收数据的操作。这8位信息的组成规定如图11.17所示。

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
EH	IR	RTS	ER	SBBK	RxE	DTR	TxEN

图11.17　命令指令的格式

（1）D_0 位为TxEN，即允许发送位，为1允许发送，为0禁止发送。

（2）D_1 位为DTR，数据终端准备好，为1时，将迫使$\overline{\text{DTR}}$输出为0。

（3）D_2 位为RxE，即允许接收位，为1允许接收，为0禁止接收。

（4）D_3 位为SBBK，发送中止字符，为1迫使TxD为低，为0正常工作。

（5）D_4 位为ER，错误标志复位，为1使PE、OE、FE复位。

（6）D_5 位为RTS，请求发送，为1迫使$\overline{\text{RTS}}$输出为低。

（7）D_6 位为IR，内部复位，为1使8251返回到方式指令格式。

（8）D_7 位为EH，同步传送时，使8251进入搜索方式，为1允许搜索同步字符。

3）状态寄存器

8251的状态寄存器用于存放8251接口的工作状态。CPU通过检查该寄存器的内容了解串行口工作状态，以便正确地执行字符入、出操作，其格式规定如图11.18所示。

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
DSR	SYNDET	FE	OE	PE	TxE	RxRDY	TxRDY

图11.18　状态寄存器的格式

（1）D_0 位TxRDY，是发送器准备好信号，输出，高有效，值为1时，表明CPU可以向8251发送数据。

（2）D_2 位TxE，是发送器为空的信号，输出，高有效，表明实现从并行到串行转换的移位寄存器已经变空，从而整个的发送器都空了。TxE与TxRDY信号的区别在于，TxE只表明数据缓冲器为空，若并行到串行转换的移位操作尚未完成，则TxE仍为低。

（3）D_1 位是RxRDY，是接收器准备好信号，输出，高有效，其为1时，表明8251已经从串行输入端接收了一个字符，则CPU可以到8251取走该字符数据。

（4）D_6 位SYNDET，只用于同步方式传送，在教学机中未使用同步传送方式，故不涉及此位。

（5）PE、OE、FE这3位为1，分别表明接口中出现奇偶错（Parity Error）、溢出错（Overrun Error）和帧错误（Framing Error）。这些错误的出现并不禁止8251继续工作。这些错误标志信号可通过命令指令中的D_4(ER)位使其复位。

2. 并行接口

串行接口按位传送数据，传送速率较低，且由于主机是按字或字节处理数据，使用串行接口时需要进行并行到串行的转换。对速度较高的设备，如打印机等，采用并行数据传送方式比较合适。我们以Intel公司的8255芯片来说明并行接口的组成和用法。

图11.19所示的Intel 8255A是一种典型的并行输入输出芯片。它有24根输入输出信

号线,在 IBM PC/AT 中,曾作为连接键盘、开关、喇叭等的接口。

尽管 CPU 可以通过往 8255A 芯片内的状态寄存器中装入命令,使它完成许多的功能,但在这里我们还是把注意力集中到其中的一些简单的操作模式上。最简单的使用 8255A 的途径是把它作为 3 个互相独立的端口 A、B 和 C,每个端口对应一个 8 位的锁存寄存器。要设置某个端口的信号,CPU 仅仅需要在数据线上发出并保持一个 8 位的数据,直到成功地改写了该端口对应的寄存器。为把某个端口用作输入端口,CPU 只需要读入对应寄存器的数据。

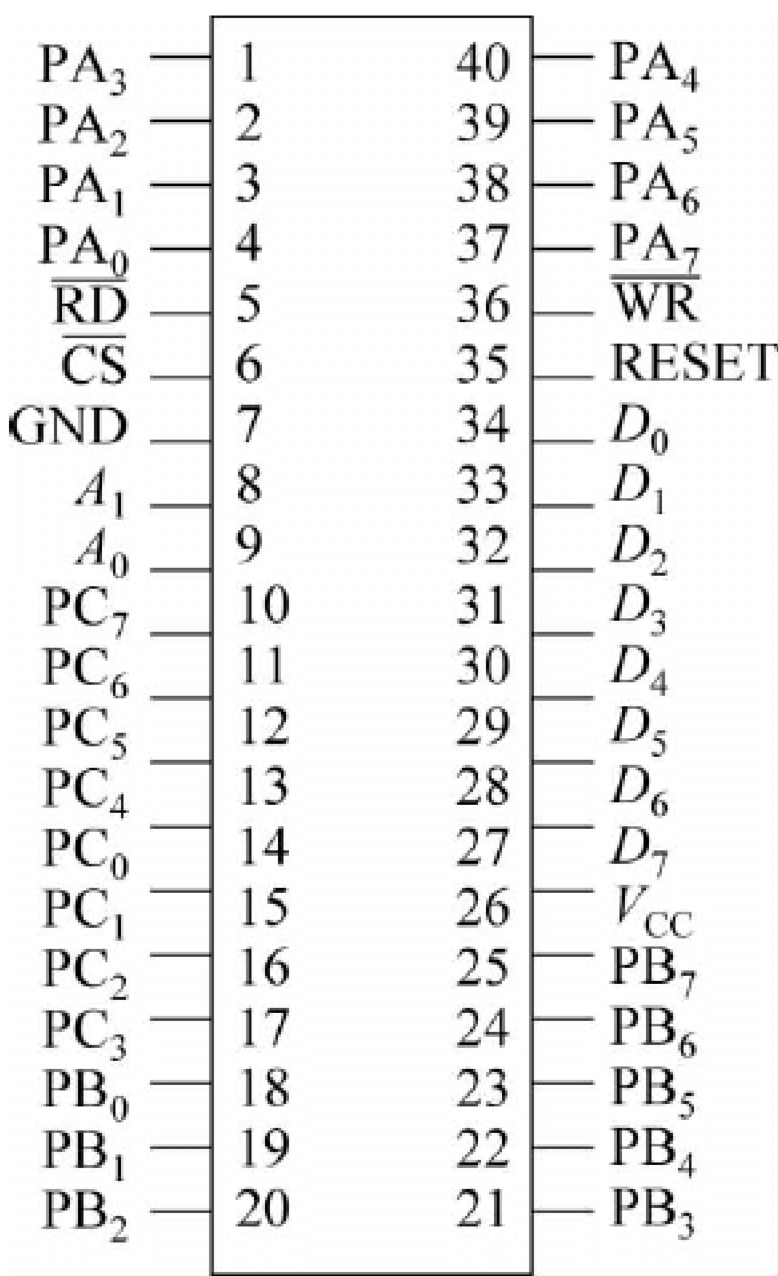


图 11.19 并行输入输出芯片 8255A

8255A 的另外一种操作模式可以用来和外部设备进行“握手”。例如,为输出数据到某个无法随时接收数据的设备,8255A 可以把数据放到某个输出端口,然后等待设备发过来的脉冲,表示设备已经接收了当前数据,并准备接收下一个数据。锁定这些脉冲信号并将其送往 CPU 所必需的逻辑电路已经包括在 8255A 的硬件中。

8255A 除了 3 个端口的 24 根信号线之外,它还有 8 根直接和数据总线相连的信号线、片选信号线、读信号线和写信号线、两根地址信号线以及用来初始化芯片的信号线。两根信号线用于选定分别对应于端口 A、B、C 和状态寄存器的 4 个内部寄存器之一。状态寄存器用来指定哪个端口用于输入,哪个端口用于输出,或者是其他功能。一般情况下,两根地址信号和地址总线的低两位连接。

11.4 输入输出方式

在计算机主机和 I/O 设备之间,可以采用不同的控制方式进行数据传送。通常分为程序直接控制方式、程序中断传送方式、直接存储器存取方式、I/O 通道控制方式和外围处理机入出方式。它们在性能、价格、致力解决问题的着重点等各方面都不一样,下面分别介绍。

11.4.1 程序直接控制方式

程序直接控制方式就是完全通过程序来控制主机和外部设备之间的信息传送。一般是在程序中安排一段由输入输出指令和其他指令组成的程序段直接控制外部设备的工作。

传送数据时,先对外部设备(接口)进行初始化,让外部设备处于就绪状态,接着 CPU 等待外部设备完成接收或发送数据的准备工作。在等待时间内,CPU 不停地用一条测试指令检测外部设备的工作状态触发器,一旦该触发器已被设置成“完成”状态,即可开始数据传送工作,一次传送完成后,循环进行下一个数据的传送。这种控制方式十分简单,但 CPU 和外部设备只能串行工作,而 CPU 的速度一般比外部设备快很多,但也只能等待外部设备处理完一次数据传送后才能进行下一次传送,系统的工作效率不高,现在仅用于和高速的外部设备之间的数据传送。

11.4.2 程序中断传送方式

程序直接控制方式中，高速的 CPU 只能在循环中等待低速的外部设备完成任务后，才能进行其他工作，系统的效率低下。如果能在 CPU 发出命令后，即可去进行其他工作，而让外部设备完成任务后，再通知 CPU 进行下一个数据的传送，则可以更好地利用 CPU，提高系统的性能。这就是程序中断传送方式。

1. 中断的基本概念

能引起中断的事件，或能发出中断请求的设备被称为中断源。按中断源的不同，可以把中断分成外中断和内中断。由各种输入输出设备、一些接口卡等引起的中断被称为外(部)中断；而由处理机硬件故障、程序运行出错等引起的中断被称为内(部)中断，例如非法指令、算术运算溢出、校验错、电源故障等都会产生内中断。

上述两类中断又被总称为硬件中断，它是针对所谓的软件中断而言的，即由写在程序中的语句(例如用户程序中的系统调用指令、trap 指令等)引起的一段程序的执行过程，它很类似于一次中断处理过程，故又称其为软件中断。这里说的软件中断是严格地与程序运行过程同步的，而硬件中断则是随机发生的。

从 CPU 要不要接收中断请求，从能不能限制某些中断发生的角度，又可以把中断分成可屏蔽中断和不可屏蔽中断。那些可以被 CPU 通过指令限制其发出中断请求(称为屏蔽中断)的中断属于可屏蔽中断，例如对某些外围设备就可以在一段时间里执行屏蔽中断，对另外一些中断是不允许执行屏蔽中断的，例如电源掉电中断，称这类中断为不可屏蔽中断。如果由于某种事件的存在，在很短的一段时间内，不允许 CPU 接收任何一个中断请求(禁止中断)，靠屏蔽全部中断是不可取的，通常是在 CPU 内部设置一个“中断允许”触发器，只有该触发器被置为“1”状态，才允许 CPU 响应中断请求，该触发器被置为“0”状态，则禁止 CPU 响应中断请求。为此，在指令系统中，要给出“开中断”指令(置“1”中断允许触发器)和“关中断”指令(清“0”中断允许触发器)。

为了管理众多的中断请求，需要按每个(类)中断处理的急迫程度，对中断进行分级管理，称其为中断优先级。在有多个中断请求时，总是首先响应与处理优先级最高的一个中断请求。如果 CPU 正在处理优先级低的一个中断，又来了优先级更高的一个中断请求，该如何处理呢？通常的处理办法是停止低优先级的中断处理过程，以便及时响应更高优先级的中断请求，在该高优先级中断处理完成之后，接下来再继续处理尚未完成的低优先级的中断，在该低优先级中断处理完成之后，返回去接着执行主程序。这种在处理中断的过程中又可以响应更高优先级中断的办法被称为中断嵌套。

2. 中断的处理过程

程序中断传送方式中，CPU 发出启动外部设备的命令后，不再等待外部设备完成任务，而是直接继续执行程序。当外部设备完成数据传送任务后，向 CPU 发出“中断请求”信号。CPU 检测到这个信号后，停止当前执行的程序，转到相应中断服务程序的入口，执行中断服务程序，完成数据传送工作，然后，再返回中断前的程序现场，继续执行原来的程序。概括来说就是一次完整的中断过程由中断请求、中断响应和中断处理 3 个阶段组成。

1) 中断请求

中断请求是由中断源发出并送给 CPU 的控制信号，由中断源设备通过置“1”设置在接

口卡上的中断触发器完成。为此,需要为每一个中断源设置一个中断触发器。如果CPU希望在一段时间内有选择地取消某个(些)中断源请求中断的权力,只要限制它(们)置“1”自己的中断触发器的能力即可,这是通过为中断源设置一个中断屏蔽触发器实现的。CPU可以按需要对它执行置“1”或清“0”的操作,中断屏蔽触发器置“1”,表示要屏蔽该设备的中断请求,即使引发中断的事件已经发生,它也不能完成置“1”自己的中断触发器的操作,仅在中断屏蔽触发器为“0”状态时(未屏蔽中断),引发中断的事件到来时才能置“1”中断触发器。

2) 中断响应

当CPU接到中断请求信号(可能多个)时,如果下面2个条件都具备,它就会响应中断请求。这2个条件包括:允许中断(允许中断触发器为“1”状态);CPU结束一条指令的执行过程,新请求的中断的优先级(比CPU此时刻正处理的任务)更高。中断响应最核心的功能是停下处于运行中的主程序的正常执行过程,准备进入中断处理阶段。这里需要插入一条没有操作码、不能供编程使用的指令(称为中断隐指令)的执行过程,或理解为一条机器指令延长出来的几个执行步骤,用于完成保存程序计数器PC的内容(主程序的下一条待执行指令的地址),或许还包括程序状态字、其他现场信息的内容到堆栈的操作功能。

3) 中断处理过程

一次中断处理过程通常要经过如下几个步骤完成,如图11.20所示。

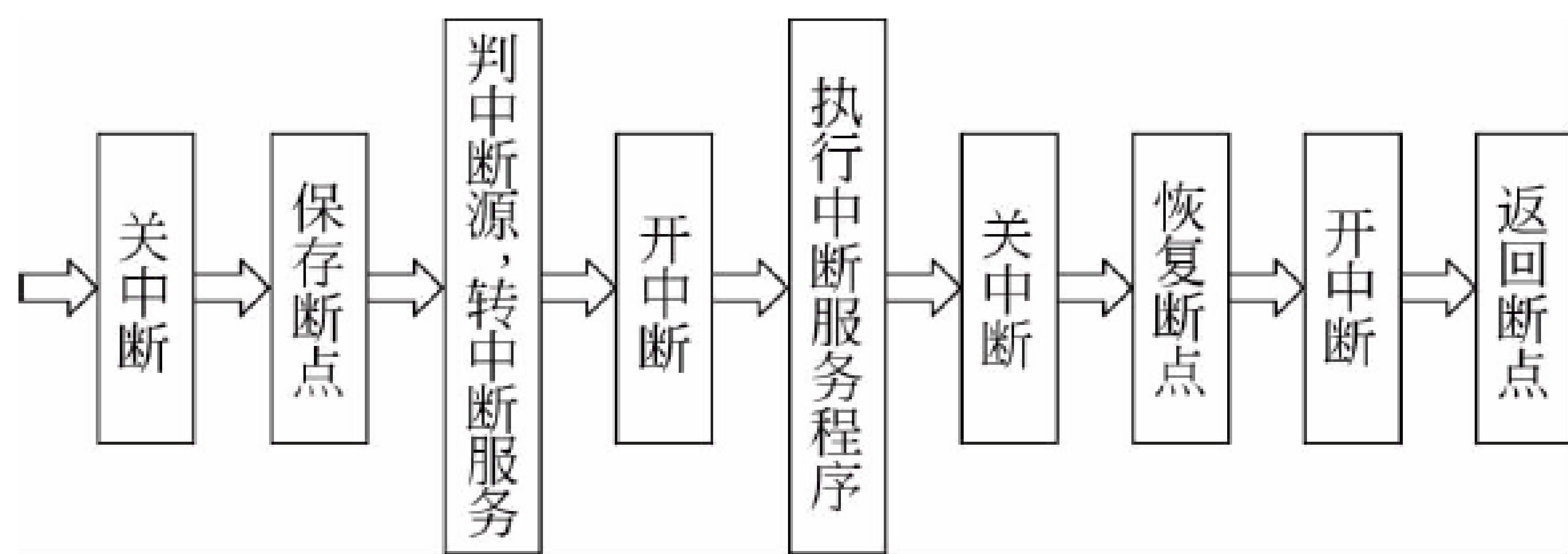


图11.20　中断处理过程

(1) 关中断,保证在此之后的一小段时间之内CPU不能响应新的中断请求。

(2) 保存断点(PC的内容,也许还包括程序状态字的内容),这一定是用中断隐指令(纯硬件机制)实现的。还要保存被停下来的程序的其他现场信息,这可以用软件实现,即到中断服务程序的开始部分完成。断点和现场信息一般保存到堆栈中去,以便支持中断嵌套;保存断点和现场信息一定要完整完成,这是中断处理完成后,保证被停下来的程序得以继续正常运行所必须的。

(3) 判别中断源,找到中断服务程序的入口地址。在多个中断源发出中断请求时,首先需要找出其中中断优先级最高的那个中断源。这可以采用硬件,也可以采用软件完成。在确定了中断源之后,接下来需要找出对应于该中断源的中断服务程序的入口地址,并将该地址传送到程序计数器PC中。得到中断服务程序的入口地址,通常有两种办法。一种在中断总控程序中用专用的INTA指令接收中断设备编码,再用该设备编码到指定的内存区中找到中断服务程序的入口地址。另外一种是中断向量法,由每个中断源直接提供中断向量,以这一中断向量为地址到中断向量表中取出中断服务程序的入口地址。中断向量表是由每个中断源的中断服务程序的入口地址组成的一张列表,通常被存放在内存中指定的一片区域中。当中断服务程序的入口地址送入PC中之后,下一条将执行的指令已经是中断服务程序的第一条指令,即已经开始中断服务处理过程。

（4）接下来应执行一条开中断指令，以便尽快地进入可以响应更高级别中断请求的运行状态，在保证程序正确执行必需的逻辑关系的前提下，把从关中断到下一次开中断之间的时间间隔设计得越短越好。

（5）若有更高级别中断请求到来，则可以进入新的中断的响应过程，否则执行中断服务程序。

（6）执行完中断服务程序，就要准备返回主程序，为此，执行关中断。

（7）接下来恢复现场信息，恢复断点。

（8）执行开中断。这里的关中断和开中断是为了保证能完整地恢复现场的操作。

（9）开中断之后，若有更高级别中断请求到来，则可以进入新的中断的响应过程；否则，返回断点进入主程序的执行过程。

程序中断传送方式在一定程度上实现了 CPU 和外部设备间的并行工作。另外，由于中断可以多层嵌套，因此，可实现 CPU 和多台外部设备并行进行数据交换。若有超过一台的外部设备发出中断请求信号，CPU 则根据各外设的中断优先级，依次进行中断响应。

这种控制方式适用于一些低速、数据传送量不大的外部设备，但对那些工作频率较高的外设，如磁盘、光盘等，数据交换是批量进行的，单位数据之间的时间间隔较短，采用程序中断控制方式还是会降低 CPU 的效率，或者丢失数据。因此，对这些设备，通常采取下面介绍的直接存储器访问方式。

11.4.3 直接存储器访问方式

直接存储器访问(Direct Memory Access，DMA)方式希望在更大的程度上将 CPU 从控制外部设备进行数据交换的工作中解脱出来，其基本思路是在外部设备和主存储器之间开辟直接的数据传送通道。DMA 方式在输入输出系统中增加了一个 DMA 控制器，需要与外部设备进行数据传送时，CPU 向 DMA 控制器发送传送数据的主存起始地址和要传送数据的数量，然后，CPU 开始执行自己的程序，DMA 控制器完成数据块的传送，给出当前传送数据的地址，取得总线的使用权，将数据发送到总线上，并计算已经发送的数据个数，直到整个数据块传送完毕，再向 CPU 发出中断信号。

在 DMA 运行方式下，高速 I/O 设备与主存储器每交换一个数据（例如一个字的内容）一般要占用一个总线周期。要交换一批数据，则可以有不同的处理方式。一是独占总线方式，从传送第一个字开始直到这批数据传输完成的整个过程，DMA 都把住总线不放，使总线只为本 DMA 使用。其缺点是 CPU 和其他 DMA 等总线主设备都要停止运行，会影响系统运行效率。二是周期挪用方式，DMA 占用总线周期传送一个字的期间，若 CPU 在此期间并不使用总线，它就继续执行指令，二者均可运行，互不干涉；若 CPU 也要使用总线，则发生了争用总线的矛盾，此时 CPU 要让出一个总线周期先给 DMA 使用，之后自己才能得到总线使用权并继续运行。

DMA 控制器是在中断接口的基础上，加上 DMA 控制逻辑构成。一般由设备识别逻辑（片选信号）、控制/状态逻辑、数据缓冲电路、中断处理逻辑、主存地址寄存器、数据数量计数器和 DMA 请求线路等部分组成。基本的 DMA 控制器如图 11.21 所示，其中 ADR 是设备地址寄存器，MAR 是主存地址寄存器，DBR 是数据缓冲寄存器，WC 是数据数量计数器，CSR 是控制状态寄存器。

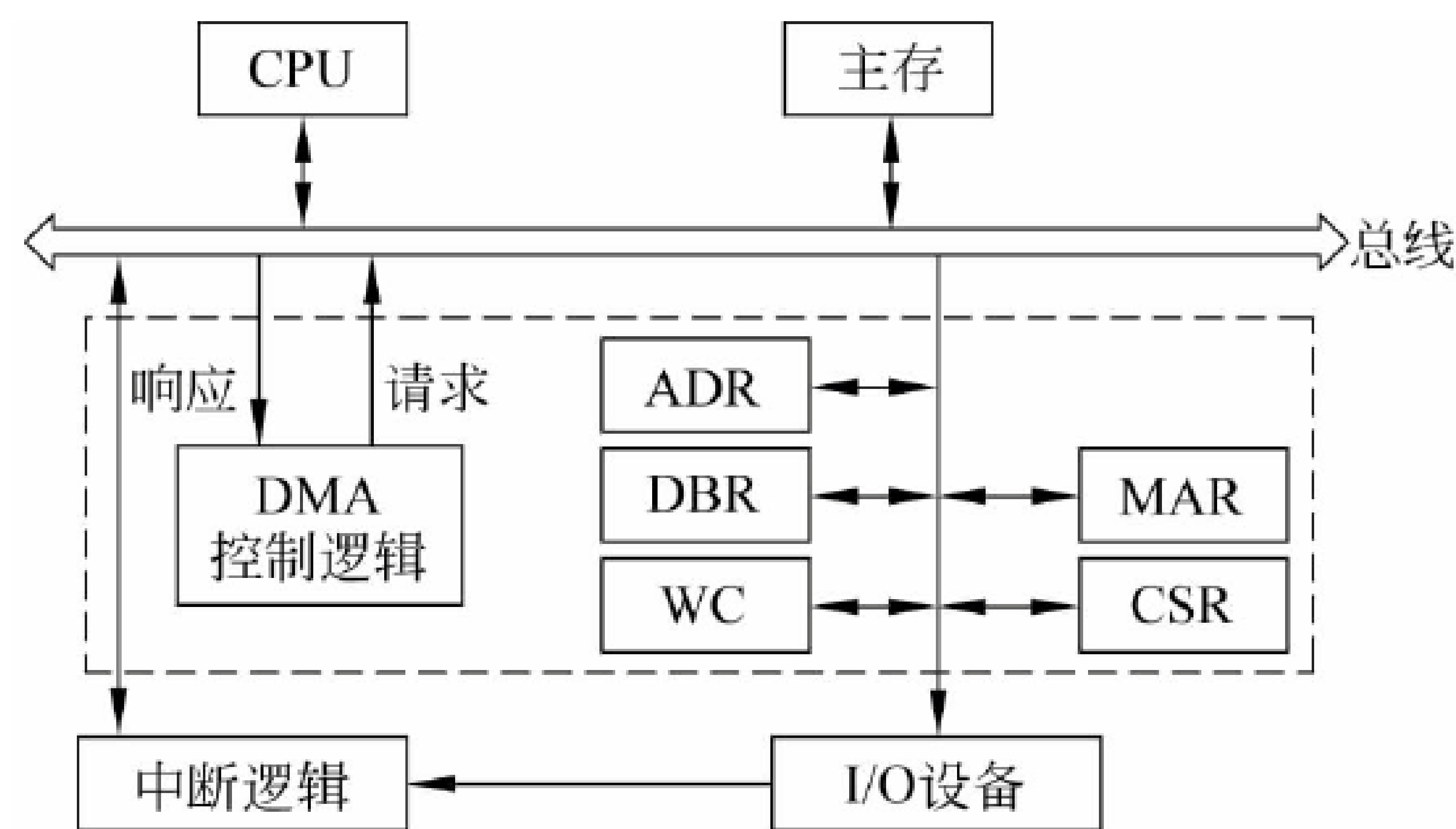


图 11.21 DMA 接口的内部组成

主存地址寄存器存放读写主存用到的主存地址，一批数据传送开始前，由 CPU 写入其初值，以后每传送一个字，该地址计数器增量，使其指向下一个主存单元。

数据数量计数器存放传送数据的数量，通常用补码给出，由 CPU 写入其初值，以后每传送一个字，该计数器加 1，当计数到 0 时，表示这批数据传送完毕，此时 DMA 应向 CPU 发中断请求信号。

DMA 的控制/状态逻辑由控制和状态等逻辑电路组成，用于修改主存地址计数器和数据数量计数器，指定传送功能(输入还是输出)，协调 CPU 和 DMA 信号的配合与同步。

DMA 控制逻辑接收并记忆设备送来的请求数据传送的信号，使其向 CPU(总线仲裁逻辑)发出 DMA 请求信号，CPU 接到这一请求信号并响应后，送回响应回答信号。DMA 的控制/状态逻辑接到这一回答信号，就取得了总线的使用权，启动数据传送，并为下一次的请求做好准备。

数据缓冲寄存器用于存放高速设备与主存之间交换的数据，也有的 DMA 卡上不设置数据缓冲寄存器，而由送出数据的一方，在指定的时刻直接把数据放到数据总线上。

中断机构与通用接口中的中断逻辑电路的组成完全相同，中断请求发生在数据数量计数器计数到 0 值的时刻，用于向 CPU 报告本组数据传送完成，并等待新的传送命令。

一次 DMA 传送过程由 3 个阶段组成，包括传送前的预处理、数据传送和传送结束处理，如图 11.22 所示。

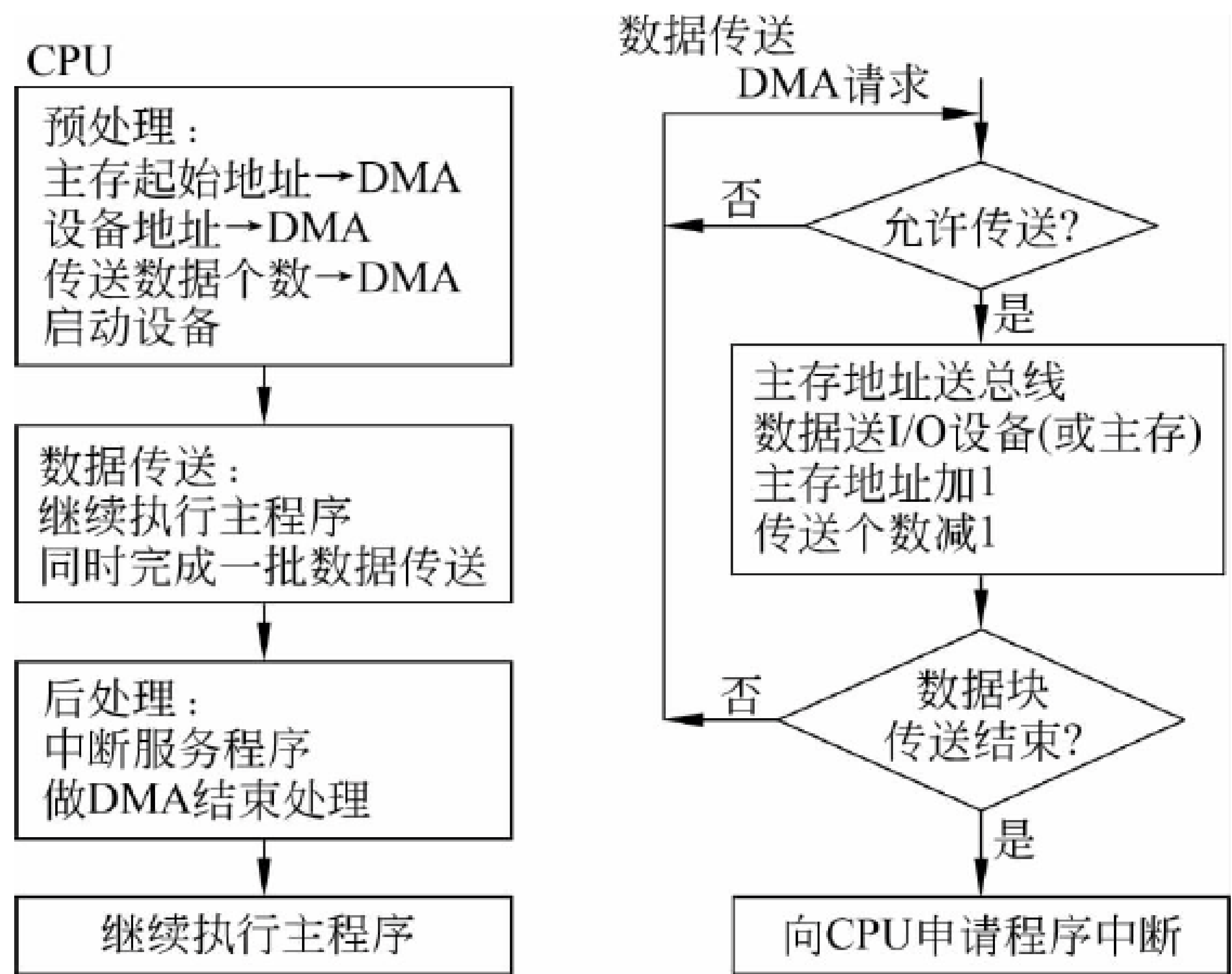


图 11.22 DMA 传送过程

传送前的预处理是由 CPU 完成的。例如,CPU 执行到读写磁盘的系统调用语句时,就要启动 DMA 传送过程,向 DMA 控制器送入设备识别信号,启动设备,测试设备运行状态,送入内存地址初值,传送数据的数据个数,DMA 的功能控制信号等。这之后,CPU 继续执行原来的程序,数据传送将在 DMA 控制器控制下,在磁盘和主存储器之间自动完成。

数据传送是在 DMA 控制器控制下自动完成的。以读磁盘为例,当磁盘准备好一个数据,它就向 DMA 控制器发出请求信号,DMA 控制器向 CPU 发出请求总线使用权的信号,若总线空闲,总线控制器将送响应回答信号给 DMA 控制器,DMA 控制器就取得了总线使用权,并启动数据传送过程,把内存地址计数器的内容送到地址总线,送一个回答信号给设备,设备就可以把准备好的一个数据送到数据总线,DMA 向内存发出命令,从而完成一次数据传送。在这个过程中,DMA 还要完成对内存地址计数器和数据数量计数器的计数操作,并通过检查数据数量计数器是否为 0,决定要启动下一次传送,还是结束本次全部数据的传送过程。

传送结束处理是由数据数量计数器的值为 0 引发出来的。数据数量计数器的值为 0 时,DMA 将向 CPU 发出中断请求信号,CPU 响应这一请求后,转入中断服务程序,检查是结束数据传送(例如响应一个磁盘读的系统调用语句全部完成),或向 DMA 发去新的操作命令,以便继续执行该系统调用语句中尚未完成的部分的传送操作。

11.4.4 I/O 通道控制方式

DMA 方式中,对外部设备的管理和一些操作的控制仍需要 CPU 承担。对大中型机来说,由于系统配备的外部设备种类多、数量也多,CPU 对外部设备管理的负担也越发繁重。I/O 通道方式就是为解决这个问题而设计的。它增强了 DMA 控制器功能,使它能独立地执行用通道命令编写的输入输出控制程序,产生相应的控制信号送给由它管辖的设备控制器,继而完成复杂的输入输出程序。也就是说,I/O 通道已具备处理器的功能,只是它的指令系统是专门为输入输出操作设计的,并在主机 CPU 的 I/O 指令指挥下,独立完成输入输出功能。

11.4.5 外围处理机方式

对大型计算机来说,它们处理的任务是复杂的海量数据,输入输出的任务更加繁重,I/O 通道控制方式可能也无法承担。这时,需要有一台或多台输入输出处理机,也称外围处理机,来承担输入输出任务、I/O 系统与设备的诊断维护以及人机交互处理等功能。外围处理机可能就是一般的小型通用计算机或微机,它不但能完成 I/O 通道所完成的 I/O 控制,还可以完成码制转换、数据格式处理、数据块的检错和纠错等操作。外围处理机可以被看作主 CPU 的伙伴和助手,促成计算机系统从完全的功能集中型向功能分布型发展。

本章内容小结和学习方法建议

当前的计算机系统中,总线的选择与使用是影响系统运行性能、特别是输入输出能力的重要因素,但总线设计与实现中会涉及许多线路与处理逻辑的细节,很复杂,远不是我们这门课程所能包括的。本章简要地介绍了计算机中的总线功能、总线结构、总线时间、总线仲

裁等,并给出了 PCI 等几种典型的计算机总线实例,要求学习过后能初步具备对总线的性能进行定性评估和定量计算的能力。

本章在综述通用可编程接口的内部组成和接口线路在计算机系统中的重要性之后,介绍了 Intel 公司的 Intel 8251 串行接口这个实例,包括其内部组成、使用方法等内容。需要能够理解接口电路在连接 CPU 和外围设备中的作用,以及需要解决的问题。

对常用的输入输出方式,重点是程序直接控制方式、程序中断方式和直接内存访问方式,给出了经常遇到的一些概念和术语。

习题与思考题

1. 计算机总线的功能是什么? 从功能区分,计算机中通常要使用哪 3 种类型的总线? 它们各自对计算机系统性能有什么影响?

2. 简单画出单总线、双总线、三总线结构的总线系统构成情况,写出各自的大体时钟频率和数据总线的位数。

3. 总线仲裁的作用是什么? 通常采用什么机制完成总线仲裁? 为什么?

4. 输入输出接口的作用是什么? 为什么不把计算机主机和输入输出设备一起设计,使它们能简单、直接地连接到一起,而要使用接口电路呢?

5. 通用可编程接口中应由哪些部件组成? 各自的功能是什么? 这里的定语“通用”和“可编程”各自的含义是什么?

6. 串行接口 Intel 8251 的通用性、可编程性表示在哪里? 为什么用两个端口地址可以访问 4 个不同的寄存器? 状态寄存器的内容是怎样被读写和使用的?

7. 计算机中有哪几种常用的输入输出控制方式? 各自的优缺点是什么?

8. 不同的输入输出方式,对 CPU 资源的占用情况有什么不同? 对 CPU 的使用效率和计算机整体性能有什么影响?

9. 中断在计算机系统中的作用有哪些?

10. 简述 CPU 可以响应中断请求的条件和时刻。

11. 开中断、关中断、中断屏蔽是指什么? 它们的作用是什么?

12. 简述一次中断处理的完整过程。

13. 某计算机有 5 级中断 $L_4 \sim L_0$,中断屏蔽字为 $M_4M_3M_2M_1M_0$,$M_i=1(0\leqslant i\leqslant 4)$表示对 L_i 级中断进行屏蔽。若中断响应优先级从高到低的顺序是 $L_0 \rightarrow L_1 \rightarrow L_2 \rightarrow L_3 \rightarrow L_4$,且要求中断处理优先级从高到低的顺序为 $L_4 \rightarrow L_0 \rightarrow L_2 \rightarrow L_1 \rightarrow L_3$,则 L_1 的中断处理程序中设置的中断屏蔽字是________。

A. 11110　　B. 01101　　C. 00011　　D. 01010

14. DMA 传输方式的优点是什么? DMA 接口中通常应包括哪些逻辑部件? 各自的功能是什么?

15. 简述 DMA 处理的完整过程。

16. DMA 控制传送一批数据,从使用总线的角度区分,有哪两种主要运行方式? 各自的优缺点是什么?

17. 综述提高计算机系统的总体的输入输出能力有哪些可行办法?

18. 假设某系统总线在一个总线周期中并行传送 4 字节信息，一个总线周期占用 2 个时钟周期，总线时钟频率为 10 MHz，则总线带宽是________。

A. 10 MB/s　　B. 20 MB/s　　C. 40 MB/s　　D. 80 MB/s

19. 在系统总线的数据线上，不可能传输的是________。

A. 指令　　B. 操作数　　C. 握手(应答)信号　　D. 中断类型号

20. 下列选项中，能引起外部中断的事件是________。

A. 键盘输入　　B. 除数为 0　　C. 浮点运算下溢　　D. 访存缺页

21. 下列选项中的英文缩写均为总线标准的是________。

A. PCI、CRT、USB、EISA　　B. ISA、CPI、VESA、EISA

C. ISA、SCSI、RAM、MIPS　　D. ISA、EISA、PCI、PCIExpress

22. 单级中断系统中，中断服务程序执行顺序是________。

Ⅰ. 保护现场　　Ⅱ. 开中断　　Ⅲ. 关中断　　Ⅳ. 保存断点

Ⅴ. 中断事件处理　　Ⅵ. 恢复现场　　Ⅶ. 中断返回

A. Ⅰ→Ⅴ→Ⅵ→Ⅱ→Ⅶ　　B. Ⅲ→Ⅰ→Ⅴ→Ⅶ

C. Ⅲ→Ⅳ→Ⅴ→Ⅵ→Ⅶ　　D. Ⅳ→Ⅰ→Ⅴ→Ⅵ→Ⅶ

23. 假定一台计算机的显示存储器用 DRAM 芯片实现，若要求显示分辨率为 1600×1200，颜色深度为 24 位，帧频为 85Hz，显存总带宽的 50%用来刷新屏幕，则需要的显存总带宽至少约为________。

A. 245Mb/s　　B. 979 Mb/s　　C. 1958 Mb/s　　D. 7834 Mb/s

24. 某计算机处理器主频为 50 MHz，采用定时查询方式控制设备 A 的 I/O，查询程序运行一次所用的时钟周期数至少为 500。在设备 A 工作期间，为保证数据不丢失，每秒需对其查询至少 200 次，则 CPU 用于设备 A 的 I/O 的时间占整个 CPU 时间的百分比至少是________。

A. 0.02%　　B. 0.05%　　C. 0.20%　　D. 0.50%

25. 某计算机的 CPU 主频为 500 MHz，CPI 为 5(即执行每条指令平均需 5 个时钟周期)。假定某外设的数据传输率为 0.5 MB/s，采用中断方式与主机进行数据传送，以 32 位为传输单位，对应的中断服务程序包含 18 条指令，中断服务的其他开销相当于 2 条指令的执行时间。请回答下列问题，要求给出计算过程。

(1) 在中断方式下，CPU 用于该外设 I/O 的时间占整个 CPU 时间的百分比是多少？

(2) 当该外设的数据传输率达到 5 MB/s 时，改用 DMA 方式传送数据。假定每次 DMA 传送块大小为 5000 B，且 DMA 预处理和后处理的总开销为 500 个时钟周期，则 CPU 用于该外设 I/O 的时间占整个 CPU 时间的百分比是多少？(假设 DMA 与 CPU 之间没用访存冲突)

第 12 章 流水线技术

为了提高处理机执行部件的处理速度,经常在计算机体系结构的部件设计中采用流水线技术。本章首先介绍流水线技术的基本概念、表示方法、流水线的特点以及流水线的分类等。在本章的第 2 部分,分析了流水线技术中用到的几个主要的性能指标。本章还简单介绍了 DLX 指令集和基本实现,并以 DLX 指令集为基础,重点讲述了 DLX 流水线的基本实现原理,对流水线中的相关问题进行了讨论,同时给出了处理各种相关的基本方法。本章最后对指令级并行技术进行了简单的介绍。

12.1 流水线的基本概念

12.1.1 流水线的概念

1. 什么是流水线

流水线技术并不是 CPU 设计和实现领域所特有的技术。在计算机还没有出现以前,工业生产过程中就已经广泛应用流水线技术,到了今天,几乎所有的现代化工厂里的生产线都采用流水线技术。例如在某一种产品的生产过程中,需要几道工序才能完成,每道工序只是完成生产过程的一小部分操作,然后送往下一道工序继续处理。如果仅就生产某一件产品从开始到完成所需要的时间来看,与不采用流水线的生产过程相比,并没有节省时间。但是从总体上来看,产品的生产率却是大大的提高了。图 12.1 给出了不采用流水线方式和采用流水线方式进行产品生产的比较示意图。

从图 12.1 中可以看到,假设生产一件产品需要 4 道工序,同样是生产 N 件产品,流水线生产方式比非流水线方式所用的时间要少得多,因为非流水线方式每 4 分钟才有一件产品生产出来,而流水线方式每隔 1 分钟就有一件产品生产出来,生产效率提高了几倍,提高的倍数是与生产工序的数目相关的,在这个例子中就是 4 倍。将上述思想引入到计算机技术中来就是处理机设计与实现的流水线技术。

计算机中的流水线是把一个重复的过程分解为若干个子过程,每个子过程与其他子过程并行进行。由于这种工作方式与工厂中的生产流水线十分相似,因此称为流水线技术,这是一种非常经济、对提高计算机的运算速度非常有效的技术,只需增加少量硬件就能把计算机的运算性能提高几倍,高的性能是通过所谓的时间并行性得到的,即不同指令的不同执行步骤是在同一个时间段完成的,下面会更详细地进行分析。

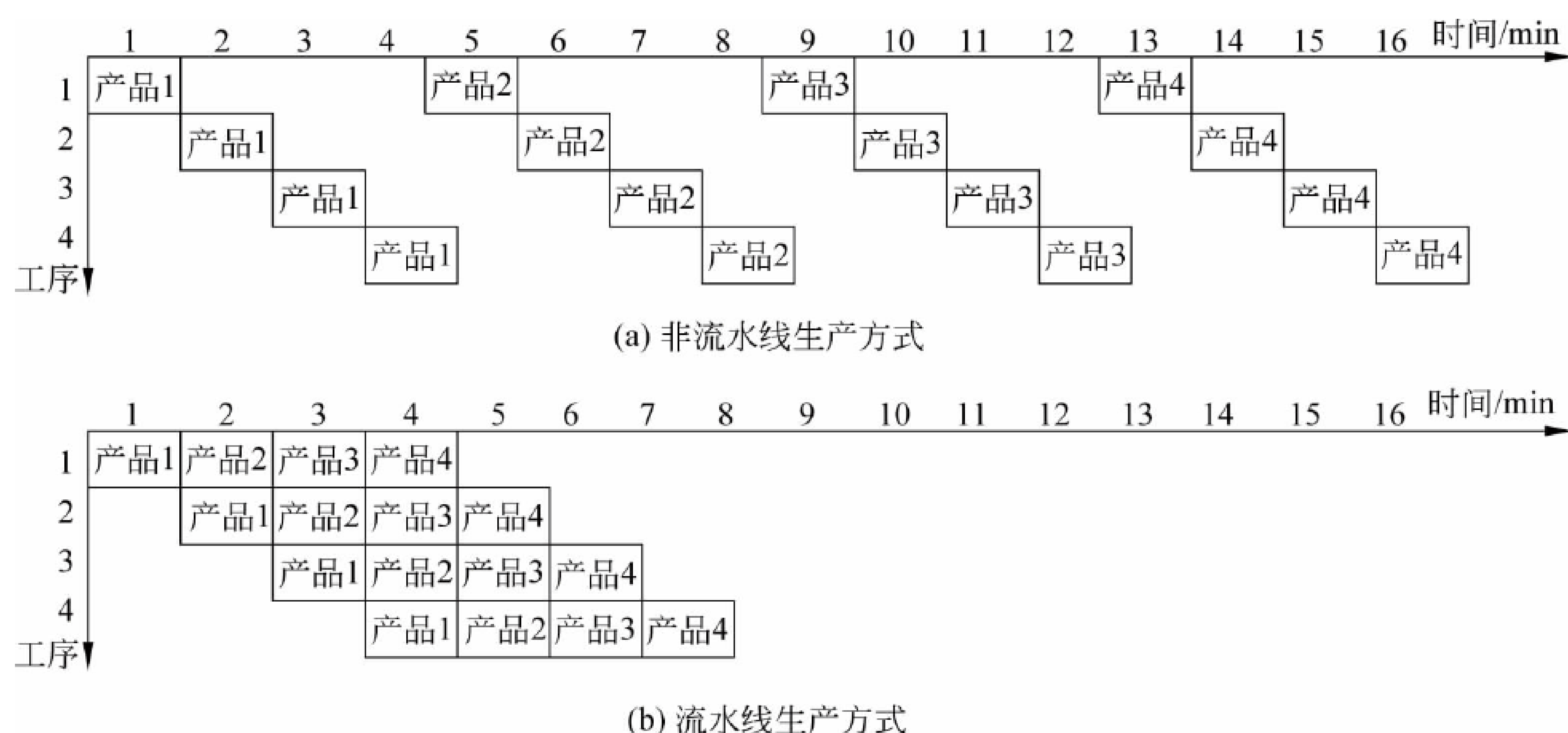


图 12.1　非流水线生产方式和流水线生产方式比较示意图

2. 指令的重叠执行

一条指令的执行过程可以分为多个阶段(或子过程),具体分法随计算机不同而不同。图 12.2 中把一条指令的执行过程分成以下 3 个阶段。



图 12.2　一条指令的执行过程

(1) 取指令。按照指令计数器的内容访问主存储器,取出一条指令送到指令寄存器。

(2) 指令分析。对指令操作码进行译码,按照给定的寻址方式和地址字段中的内容形成操作数的地址,并用这个地址读取操作数。

(3) 指令执行。根据操作码的要求,完成指令规定的功能,即把运算结果写到通用寄存器或主存中。

当多条指令在处理机中执行时,可以采用以下几种方式。

1) 顺序执行方式

指令的执行过程如图 12.3(a)所示。顺序执行方式执行 n 条指令所用的时间为

$$T = \sum_{i=1}^{n}(t_{\text{取指}i} + t_{\text{分析}i} + t_{\text{执行}i}) \tag{12.1}$$

如果取指令、分析指令、执行指令的时间都相等,每段的时间都为 t,则 n 条指令所用的时间为

$$T = 3nt \tag{12.2}$$

传统的冯·诺依曼机器采用顺序执行方式,又称为串行执行方式,优点是控制简单,节省设备。主要的缺点有:一是处理机执行指令的速度很慢,只有当上一条指令全部执行完毕后下一条指令才能够开始执行。二是功能部件的利用率很低,如取指令时主存是忙碌的,而指令执行部件是空闲的。而执行指令时指令执行部件是忙碌的,而主存经常是空闲的。

2) 一次重叠执行方式

这种方式把执行第 k 条指令与取第 $k+1$ 条指令同时进行,如图 12.3(b)所示。如果执行一条指令的 3 个阶段时间均相等,则执行 n 条指令所用的时间为

$$T = (1 + 2n)t \tag{12.3}$$

采用一次重叠执行方式后带来了两个优点:一是程序的执行时间缩短了近一半;二是

功能部件的利用率明显提高。主存基本上可以处于忙碌状态,其他功能部件的利用率也得到提高。为此需要付出一定的代价,即需要增加一些硬件,控制过程也变得复杂一些。

3) 二次重叠执行方式

为了进一步提高指令的执行速度,可以把取 $k+1$ 条指令提前到分析第 k 条指令同时进行,而将分析第 $k+1$ 条指令与执行第 k 条指令同时进行,如图 12.3(c)所示。如果执行一条指令的 3 个阶段时间均相等,则执行 n 条指令的所用时间为

$$T = (n+2)t \tag{12.4}$$

与顺序执行方式相比,采用二次重叠执行方式能够使指令的执行时间缩短近 2/3。这是一种理想的指令执行方式,在正常情况下,处理机中同时有 3 条指令在执行。

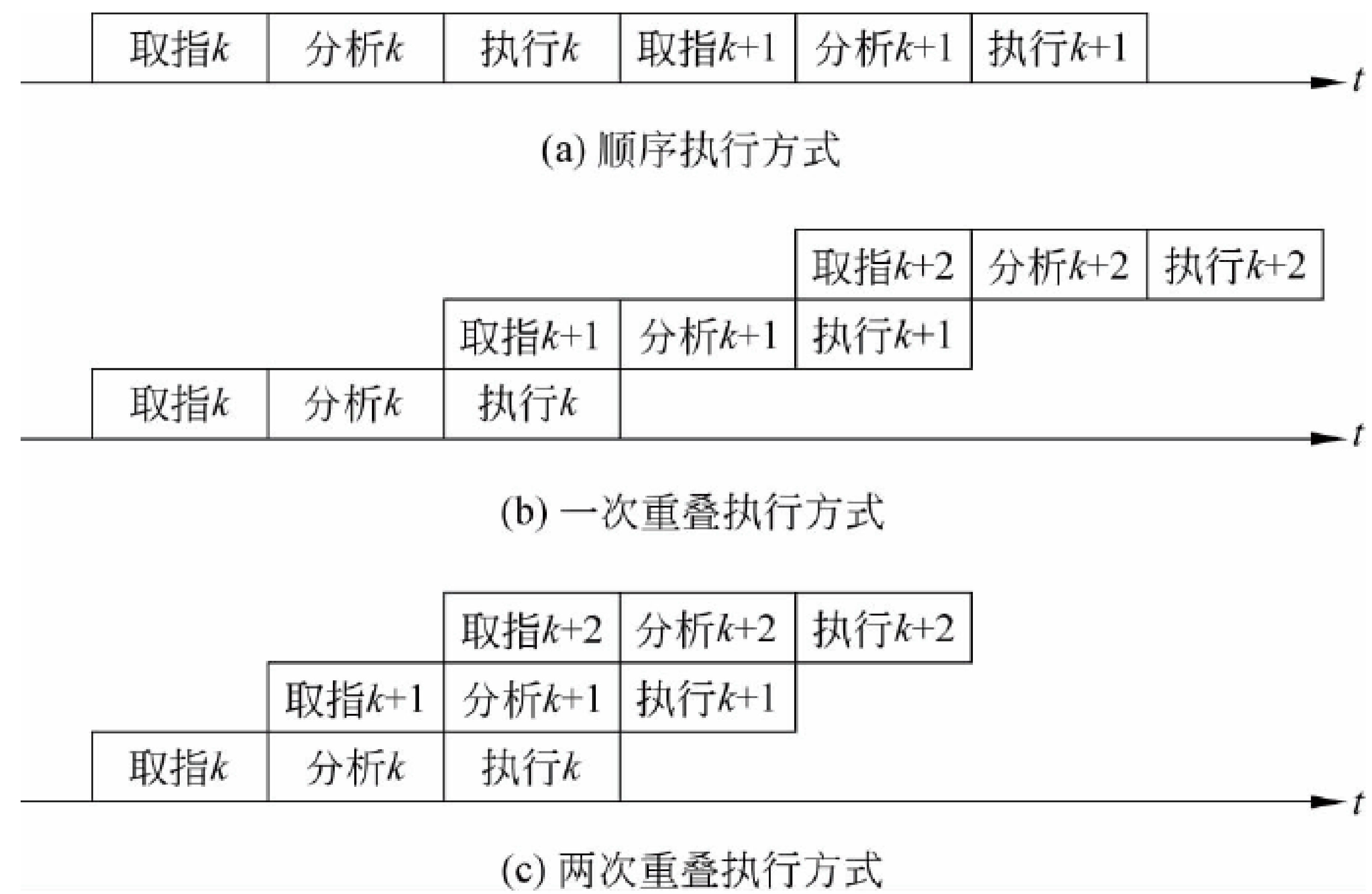


图 12.3 指令的几种执行方式

上面介绍的指令重叠执行方式实际上就是指令流水线,指令流水线是多条指令并行执行的一种实现技术。

12.1.2 流水线的表示方法

在计算机的流水线中,流水线的每一个阶段完成一条指令的一部分,不同阶段并行完成流水线中不同指令的不同部分。流水线中的每一个阶段称为一个流水阶段、流水节拍、流水步、流水段等。一个流水阶段与另一个流水阶段相连接形成流水线。指令从流水线的一端进入,经过流水线的处理,从另一端流出。

图 12.4 是一种流水线的连接图表示方法。这里一条指令的执行过程分为取指令、译码、执行、保存结果 4 个流水段。目前,大部分处理机的指令流水线在 3~12 段之间。

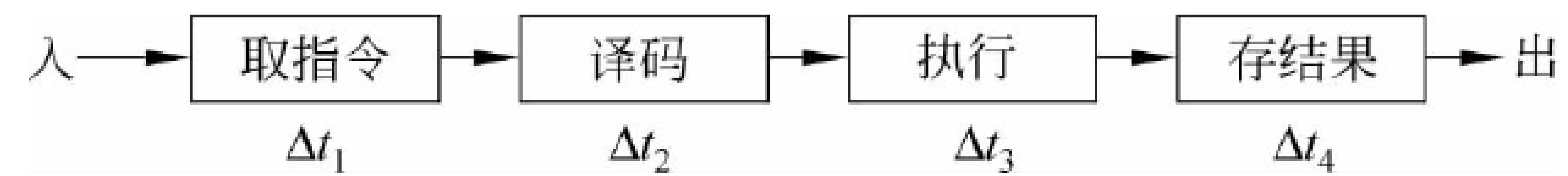


图 12.4 4 段指令流水线

有些复杂的指令在执行阶段也采用流水线方式工作,这种流水线称为操作流水线或功能部件流水线。图 12.5 是一个浮点加法器的 4 段流水线,它将浮点加法的执行过程分解为

求阶差、对阶、尾数加、规格化4个子过程，每一个子过程可以在各自独立的功能部件上完成。

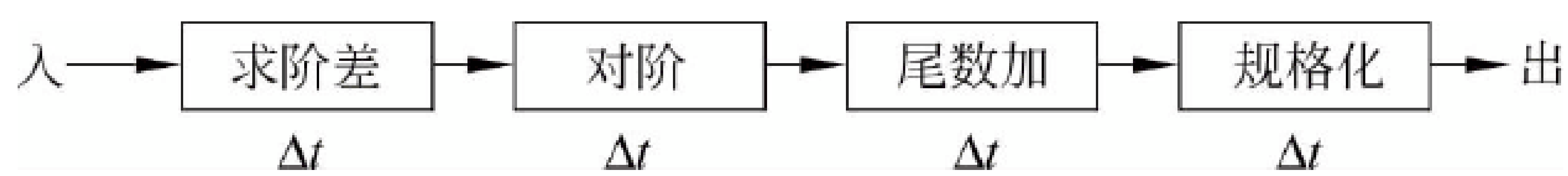


图12.5　浮点加法器流水线

在图12.5中，各个部件的执行时间都是Δt，虽然执行一次浮点加法的时间仍需要$4\Delta t$，然而由于4个部件同时工作，每隔一个Δt就能够完成一次浮点加法，输出一个运算结果。因此，采用4级流水线的浮点加法器，处理机执行浮点加法的速度能提高3倍。

为了直观描述流水线的工作过程，最常用的一种方法是采用时空图。例如图12.4所示的指令流水线，当采用时空图表示时，如图12.6所示。

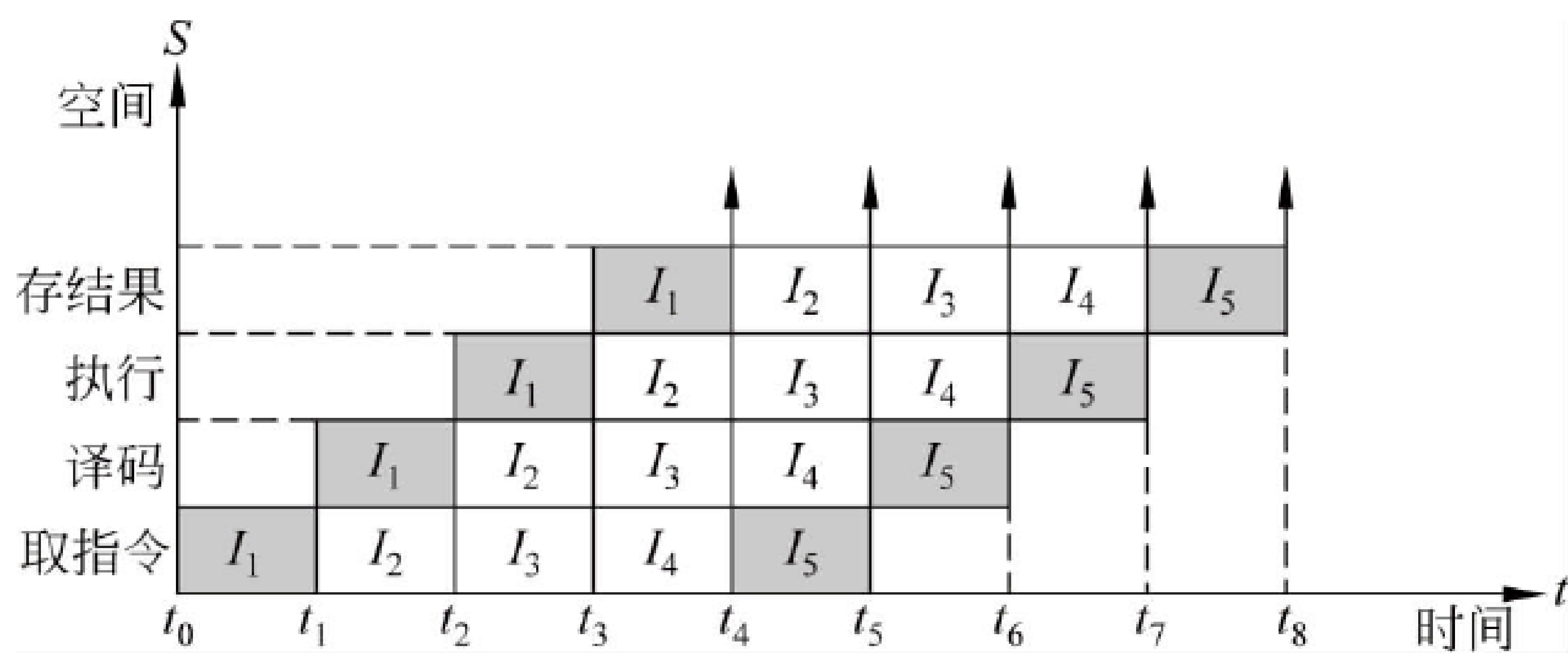


图12.6　指令流水线时空图

在时空图中，横坐标表示时间，也就是输入到流水线中的各个任务在流水线中所经过的时间。当流水线中各个流水段的执行时间都相等时，横坐标被分割成相等长度的时间段。纵坐标表示空间，即流水线的每一个流水段。图12.6中，第1条指令I_1在t_0时刻进入流水线，在t_4时刻流出流水线。第2条指令在t_1时刻进入流水线，在t_5时刻流出流水线。依此类推，每经过一个Δt时间，便有一条指令进入流水线，同时也有一条指令流出流水线。由图12.6看出，当$t_8=8\Delta t$时，流水线上便有5条指令输出。

如果不采用流水方式，而采用串行方式执行指令，如图12.6中用网点表示的I_1和I_5指令的组合情况，当$t_8=8\Delta t$时，只执行了2条指令。可见流水方式成倍提高了计算机的运行效率。

12.1.3　流水线的特点

从上面的分析可以看到，计算机中采用流水线方式与采用传统的串行方式相比较，具有如下特点。

第一，把一个任务(一条指令或一个操作)分解为几个有联系的子任务，每个子任务由一个专门的功能部件来实现。所以，流水线实际上是把一个大的功能部件分解为多个独立的功能部件，并依靠多个功能部件并行工作来缩短程序的执行时间。

第二，流水线每一个功能段部件后面都要有一个缓冲寄存器(锁存器)，其作用是保存本流水段的结果，如图12.7所示。由于流水线中每一个流水段的延迟时间不可能绝对相等，再加上电路的延迟时间和时钟等都存在偏移，因此流水段之间传送任务时，必须通过锁存器。

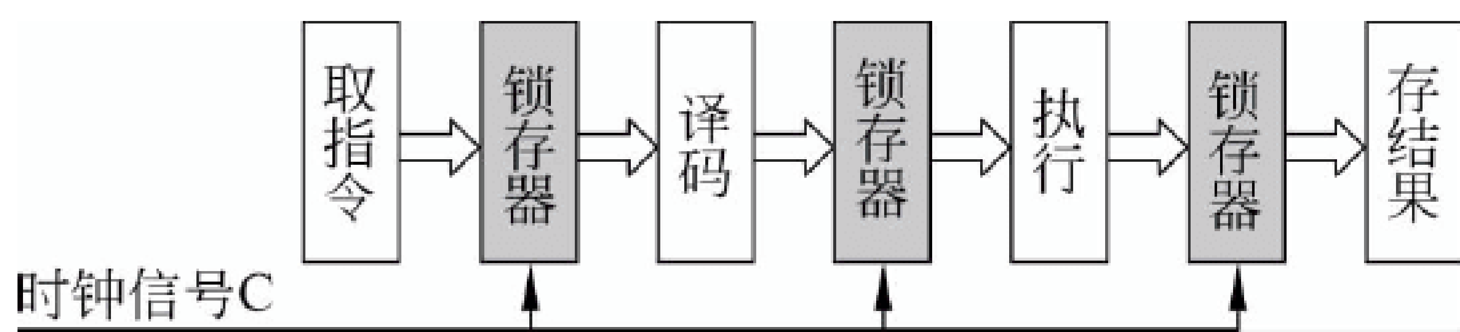


图 12.7 流水线中的锁存器

第三，流水线中各功能段的延迟时间应尽量相等，否则将引起堵塞、断流。执行时间长的一个流水段将成为整个流水线的瓶颈，流水线中的其他功能部件不能充分发挥作用。

第四，只有连续不断地提供同一种任务时才能发挥流水线的效率，即流水线中处理的必须是连续任务。在采用流水线方式工作的处理机中，特别是当流水线的级数较多时，要在软件和硬件设计等多方面尽量为流水线提供连续的任务，才可能提高流水线的效率。

第五，流水线需要有装入时间和排空时间。装入时间是指第一个任务进入流水线到输出流水线的时间。排空时间是指最后一个任务进入流水线到输出流水线的时间。在这两种情况下，流水线不满载。只有流水线完全满载时，整个流水线的效率才能得到充分发挥。

12.1.4 流水线的分类方法

根据不同的分类标准，可以把流水线分成多种不同的种类。平时所说的某种流水线，都是按照某种观点，或者从某个特定角度对流水线进行分类的结果。下面就从几个不同的角度介绍一下流水线的基本分类方法。

1. 部件功能级、处理机级和处理机之间级流水线

根据使用流水线的级别差异，可以把流水线分为部件功能级、处理机级和处理机之间级等多种流水线类型。

(1) 所谓功能部件级流水线也可以称为运算操作流水线(Arithmetic Pipelines)。图12.5中的浮点加法器就是一种典型的功能部件级流水线。要提高执行部件执行算术逻辑运算操作的速度，除了在运算操作部件中采用流水线之外，还可以设置多个独立的操作部件，并通过这些操作部件的并行工作来提高处理机执行算术逻辑运算的速度。通常，把指令执行部件中采用了流水线的处理机称为流水线处理机或超流水线处理机；而把指令执行部件中设置有多个操作部件的处理机称为多操作部件处理机或超标量计算机。

(2) 所谓处理机级流水线，又叫指令流水线(Instruction Pipelines)，它是把执行指令的过程按照流水方式处理，使处理机能够重叠地执行多条指令，即把一条指令的执行过程分解为多个子过程，每个子过程在一个独立的功能部件中完成。本章将重点介绍指令流水线。

(3) 所谓处理机之间流水线，又被称为宏流水线(Macro Pipelines)。图12.8是一种宏流水线的示意图，这种流水线由两个或者两个以上的处理机通过存储器串行连接起来，每个处理机对同一数据流的不同部分分别进行处理，前一个处理机的输出结果存入存储器中，作为后一个处理机的输入，每个处理机完成整个任务的一部分。这一般属于异构型多处理机系统，它对提高各个处理机的效率有很大的作用。

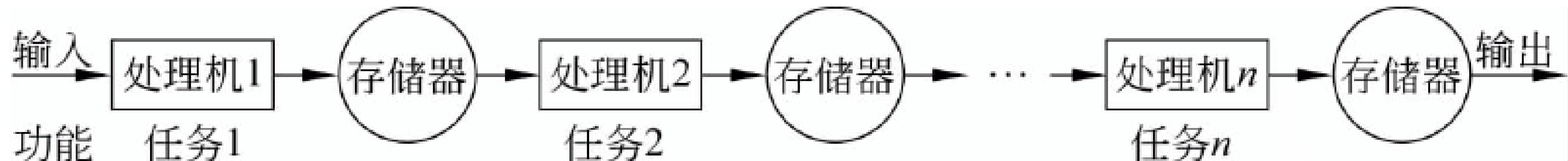


图 12.8 一种宏流水线

2. 单功能流水线和多功能流水线

根据流水线能够完成的功能来分类,可以将流水线分成单功能流水线和多功能流水线。

(1) 如果一条流水线只能完成一种固定的功能,这种流水线称为单功能流水线(Unifunction Pipelines)。例如前面介绍的浮点加法器流水线专门完成浮点加法运算。当要完成多种不同功能时,可以采用多条单功能流水线。如 Pentium 处理机有 2 条 5 段的 32 位整数运算流水线和一条 8 段的浮点运算流水线。

(2) 多功能流水线(Multifunction Pipelines)是指流水线的各段可以进行不同的连接。在不同时间内,或者在同一时间内,通过不同的连接方式实现不同的功能。多功能流水线的典型代表是 Texas 公司 ASC 中采用的 8 段流水线。在一台 ASC 处理机内有 4 条相同的流水线,每条流水线通过不同的连接方式可以完成整数加减法运算、整数乘除法运算、浮点加法运算和浮点乘法运算等功能。图 12.9 中给出了 TI-ASC 的流水线分段示意图和实现两种不同功能的连接方式。

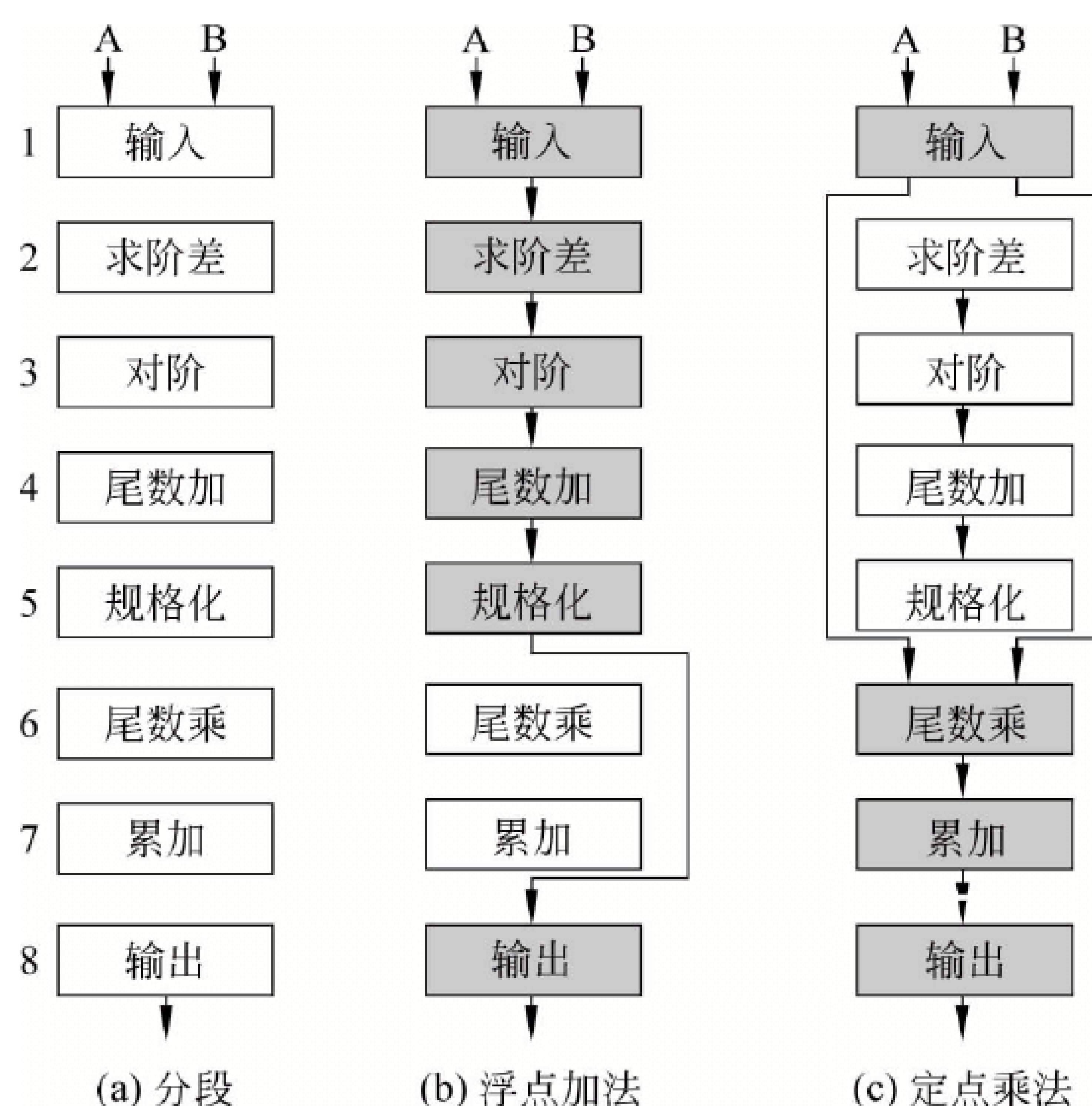


图 12.9　TI-ASC 计算机的多功能流水线

3. 静态流水线和动态流水线

在多功能流水线中,按照在同一时间内是否能够连接成多种方式,同时执行多种功能,可以把多功能流水线分为静态流水线和动态流水线两种。

(1) 静态流水线(Static Pipelines)是指在同一段时间内,多功能流水线中的各个功能段只能够按照一种固定的方式连接,实现一种固定的功能。只有当按照这种连接方式工作的所有任务都流出流水线之后,多功能流水线才能重新进行连接,从而实现其他功能。

(2) 动态流水线(Dynamic Pipelines)是指在同一段时间内,多功能流水线中的各段可以按照不同的方式连接,同时执行多种功能。这种同时实现多种连接方式是有条件的,即流水线中的各个功能部件之间不能发生冲突。

前面介绍的 TI-ASC 的 8 段流水线,如果按照图 12.10(a)中的时空图工作,就是一种静态流水线。开始时,多功能流水线按照实现浮点加减法的方式连接,当 n 个浮点加减法全部

执行完成,而且最后一个浮点加减法运算的排空操作也做完之后,多功能流水线才重新开始按照实现定点乘法的方式进行连接,并开始做定点乘法运算。如果按照图12.10(b)中的时空图工作,就是一种动态流水线。如图12.10(b)所示,中间有一段时间,在同一条多功能流水线的不同功能段中同时执行浮点加减法和定点乘法两种运算,即在浮点加减法运算还没有全部完成的情况下,定点乘法运算就已经开始了。两种运算同时在同一条多功能流水线中分别使用不同的功能段。

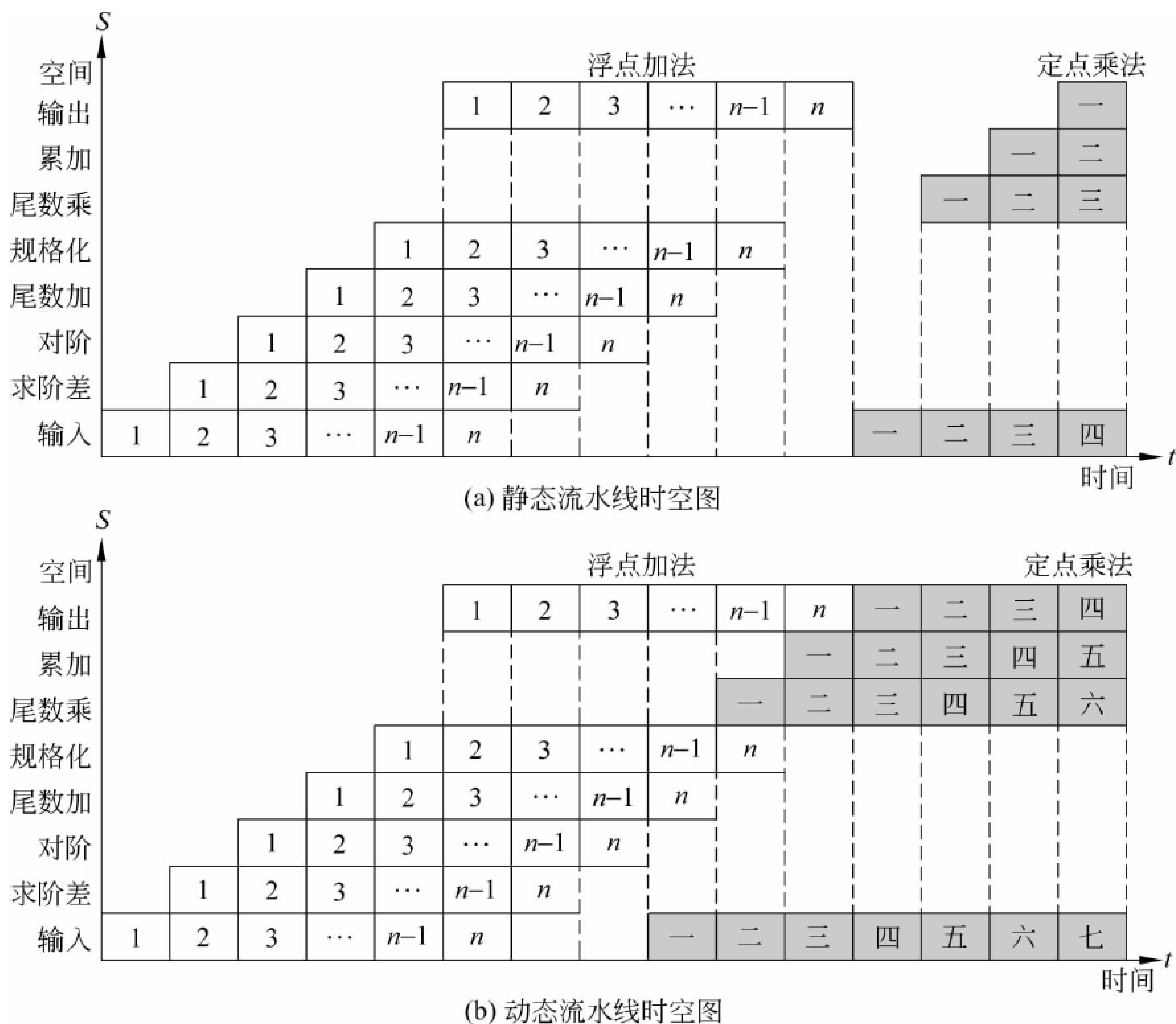


图12.10　静态流水线和动态流水线

在静态流水线中,只有程序中连续出现同一种运算时,流水线的效率才能充分发挥出来。如果输入到流水线的是一串不同运算相互间隔的操作,例如输入的是浮点加、定点乘、浮点加、定点乘……,那么这条静态流水线的效率就与顺序执行方式没有区别了。而动态流水线则不同,它允许两种运算在同一条流水线中同时执行。因此,在一般情况下,动态流水线的效率和功能部件的利用率比静态流水线高。当然,动态流水线的控制比静态流水线也要复杂得多。目前,大多数处理机中采用的都是静态流水线。

4. 线性流水线和非线性流水线

根据流水线的各个功能段之间是否有反馈信号,可以把流水线分为线性流水线和非线性流水线两类。

(1) 线性流水线(Linear Pipelines)是将流水线的各段串行连接起来,没有反馈回路。输入数据从流水线的一端进入,从另一端输出。数据在流水线的各个功能段流过时,每个功能段都流过且仅流过一次。前面例子中提到的基本上都是线性流水线。

（2）非线性流水线（Nonlinear Pipelines）则是在流水线的各个功能段之间除了有串行的连接之外，还有反馈回路。图12.11就是一个简单的非线性流水线。虽然它由4段S_1～S_4组成，但由于反馈回路的存在，在一次流水过程中，有的段可能要多次被使用，比如从输入到输出就可能依次流过S_1、S_2、S_3、S_4、S_2、S_3各段，S_2、S_3就分别被使用了两次。

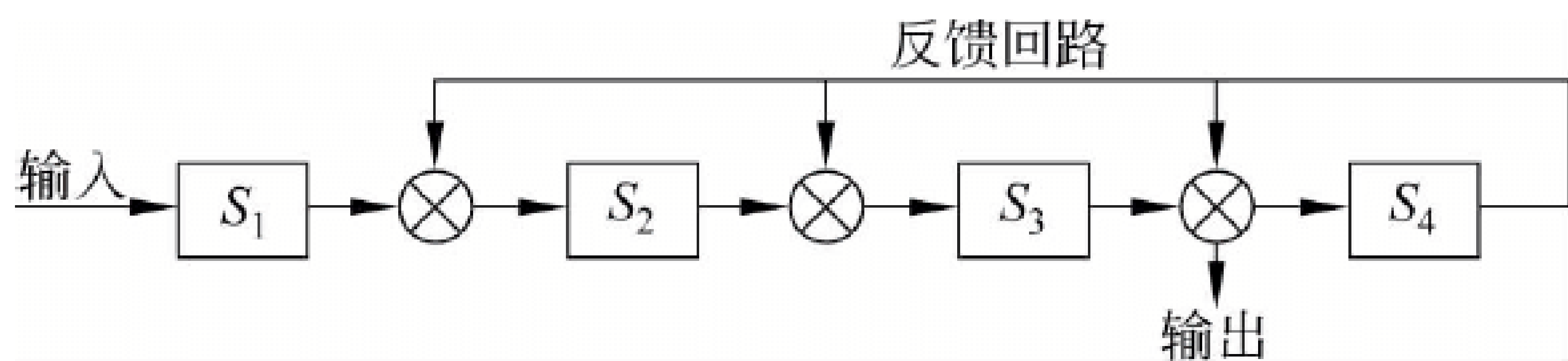


图12.11　非线性流水线

由于非线性流水线的特点，它经常用于递归调用或者构成多功能流水线等。非线性流水线中的一个重要问题是要确定在什么时间可以向流水线中输入新的任务，使新输入任务与流水线中原来的反馈任务之间在各个功能段上都不产生冲突。这就是非线性流水线的调度问题，这里就不详细介绍了。

12.2　流水线的性能指标

衡量流水线性能的主要指标有吞吐率、加速比和效率。另外，在流水线设计中，流水线的最佳段数的选择也是一个重要问题。下面就以线性流水线为例，分析流水线的主要性能指标。其分析方法和有关公式也适用于非线性流水线。

12.2.1　流水线的吞吐率

吞吐率（Throughput Rate）是衡量流水线速度的重要指标。它是指在单位时间内流水线所完成的任务数量，或是输出结果的数量，计算流水线吞吐率的最基本的公式表示为

$$\mathrm{TP} = \frac{n}{T_k} \tag{12.5}$$

式中，n为任务数，T_k为处理完成n个任务所用的时间。下面就根据流水线中各段执行时间是否相等分别对流水线的吞吐率进行讨论。

1. 各段执行时间均相等的流水线

图12.12是各段执行时间均相等的流水线时空图。当输入到流水线中的任务是连续的理想情况，一条k段线性流水线能够在$k+n-1$个时钟周期内完成n个任务。

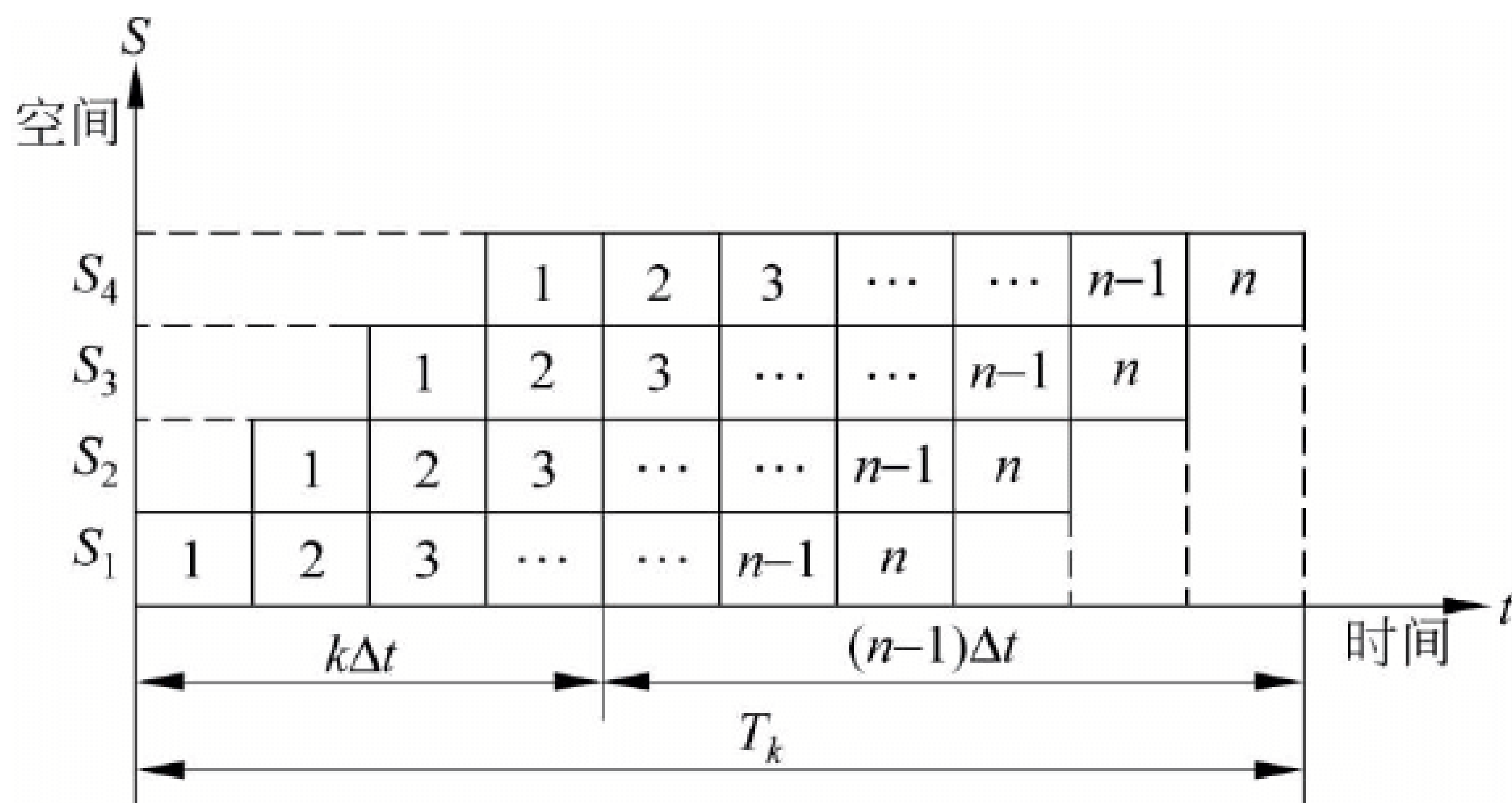


图12.12　各段执行时间均等的流水线时空图

从流水线输出端看,用 k 个时钟周期输出第一个任务,其余 $n-1$ 个时钟周期,每个时钟周期输出一个任务,即用 $n-1$ 个时钟周期输出 $n-1$ 个任务。因此,流水线完成 n 个连续任务需要的总时间为

$$T_k = (k + n - 1)\Delta t \tag{12.6}$$

式中,k 为流水线的段数,Δt 为时钟周期。将式(12.6)代入式(12.5)中,得流水线的实际吞吐率为

$$\mathrm{TP} = \frac{n}{(k + n - 1)\Delta t} \tag{12.7}$$

当连续输入的任务 $n \to \infty$时,得最大吞吐率为

$$\mathrm{TP}_{\max} = \lim_{n \to \infty} \frac{n}{(k + n - 1)\Delta t} = \frac{1}{\Delta t} \tag{12.8}$$

毫无疑问,流水线的实际吞吐率要小于最大吞吐率,它除了与时钟周期 Δt 有关外,还与流水线的段数 k、输入到流水线中的任务数 n 等有关。只有当 $n>>k$ 时,才有 $\mathrm{TP} \approx \mathrm{TP}_{\max}$。

2. 各段执行时间不相等的流水线

图 12.13(a)表示各段执行时间不相等的流水线,其中第 2 段的执行时间是其他各段执行时间的 3 倍。在这种情况下,流水线的时空图如图 12.13(b)所示。其中 S_1、S_3、S_4各段中的灰色部分表示该段流水线在这一段时间内是空闲的,而 S_2段没有任何空闲,因此 S_2段成为瓶颈段。

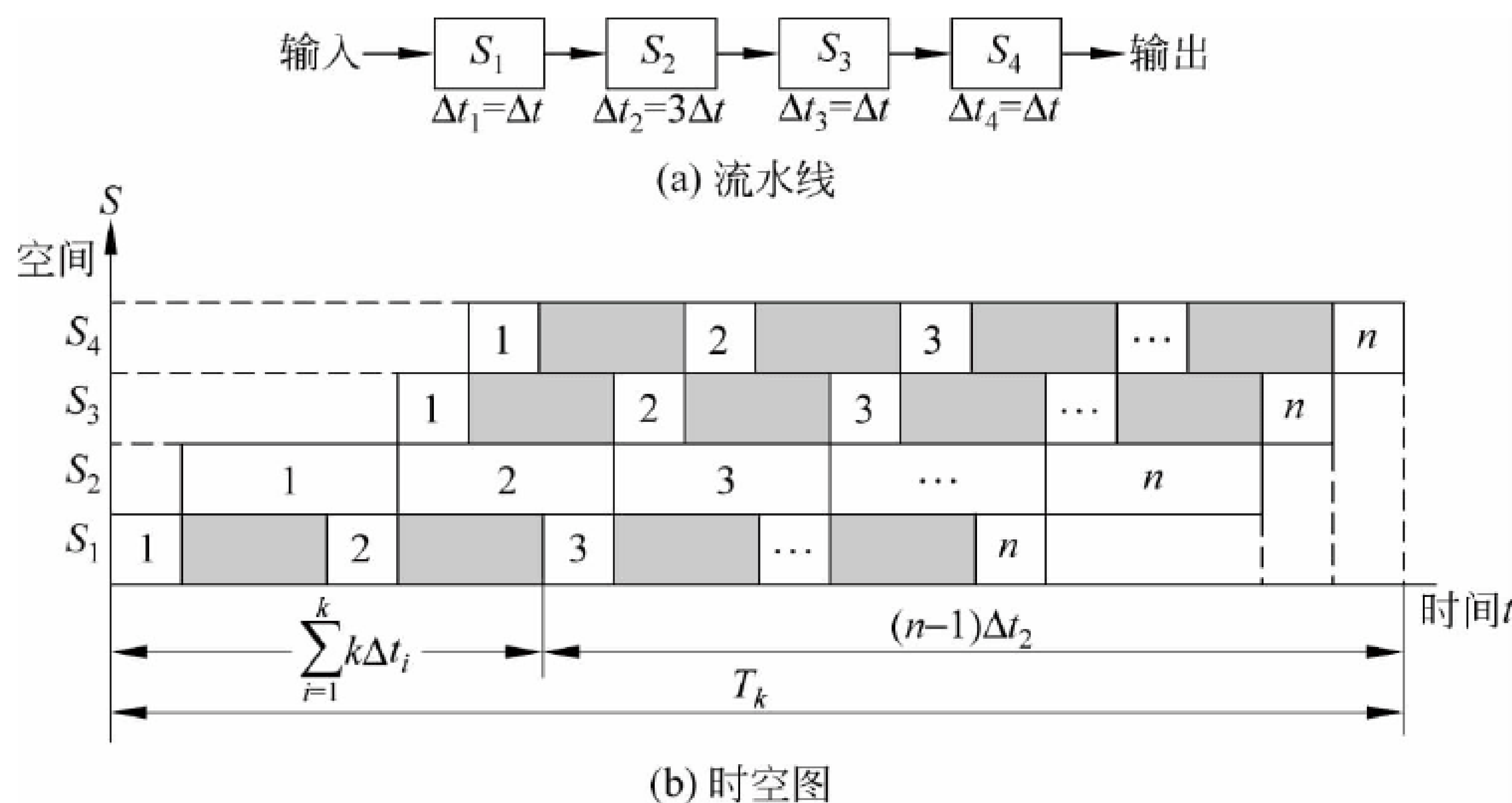


图 12.13　各段执行时不相等的流水线时空图

流水线存在瓶颈段的情况下,实际吞吐率为

$$\mathrm{TP} = \frac{n}{\sum_{i=1}^{k} t_i + (n-1)\max(\Delta t_1, \Delta t_2, \cdots, \Delta t_k)} \tag{12.9}$$

分母中的第一部分是流水线完成第 1 个任务所用时间,第二部分是完成其余 $n-1$ 个任务所用时间。此时流水线的最大吞吐率为

$$\mathrm{TP}_{\max} = \frac{1}{\max(\Delta t_1, \Delta t_2, \cdots, \Delta t_k)} \tag{12.10}$$

对于图 12.13 所示的例子,流水线的最大吞吐率为

$$\mathrm{TP}_{\max} = \frac{1}{3\Delta t} \tag{12.11}$$

从上述表达式看出，当流水线中各流水段执行时间不相等时，流水线的最大吞吐率与实际吞吐率主要是由执行时间最长的那个流水段决定。这个流水段就成了整个流水线的瓶颈。此时，瓶颈流水段一直处于忙碌状态，而其余各段有许多空闲时间，这是一种资源浪费。

解决流水线瓶颈问题有 2 种可行方法。一种方法是将流水线的瓶颈部分再细分，如图 12.14 所示，即把第 2 个流水段再划分为 3 个子流水段，分别命名为 S_{2a}、S_{2b}、S_{2c}，使每一个流水段和子流水段的执行时间均为Δt。

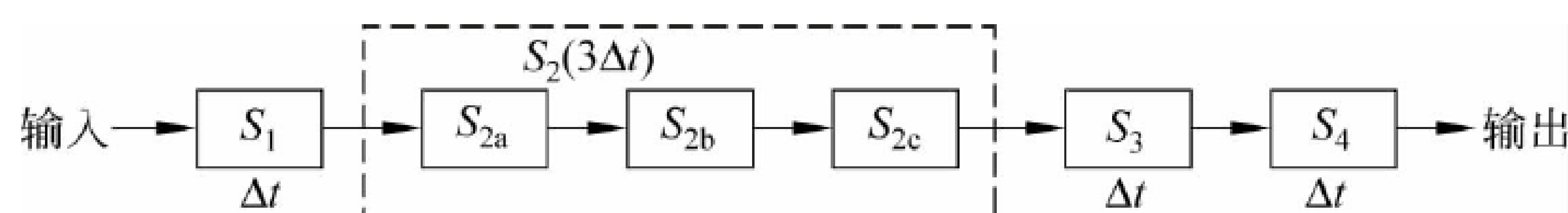


图 12.14　瓶颈流水段再次细分

第二种方法：由于结构等方面的原因，瓶颈段不能再细分时，可通过重复设置瓶颈流水段，让多个瓶颈流水段并行工作。如图 12.15(a)表示流水线连接图，图 12.15(b)是流水线时空图。两种流水线的最大吞吐率同样为

$$\mathrm{TP}_{\max} = \frac{1}{\Delta t} \tag{12.12}$$

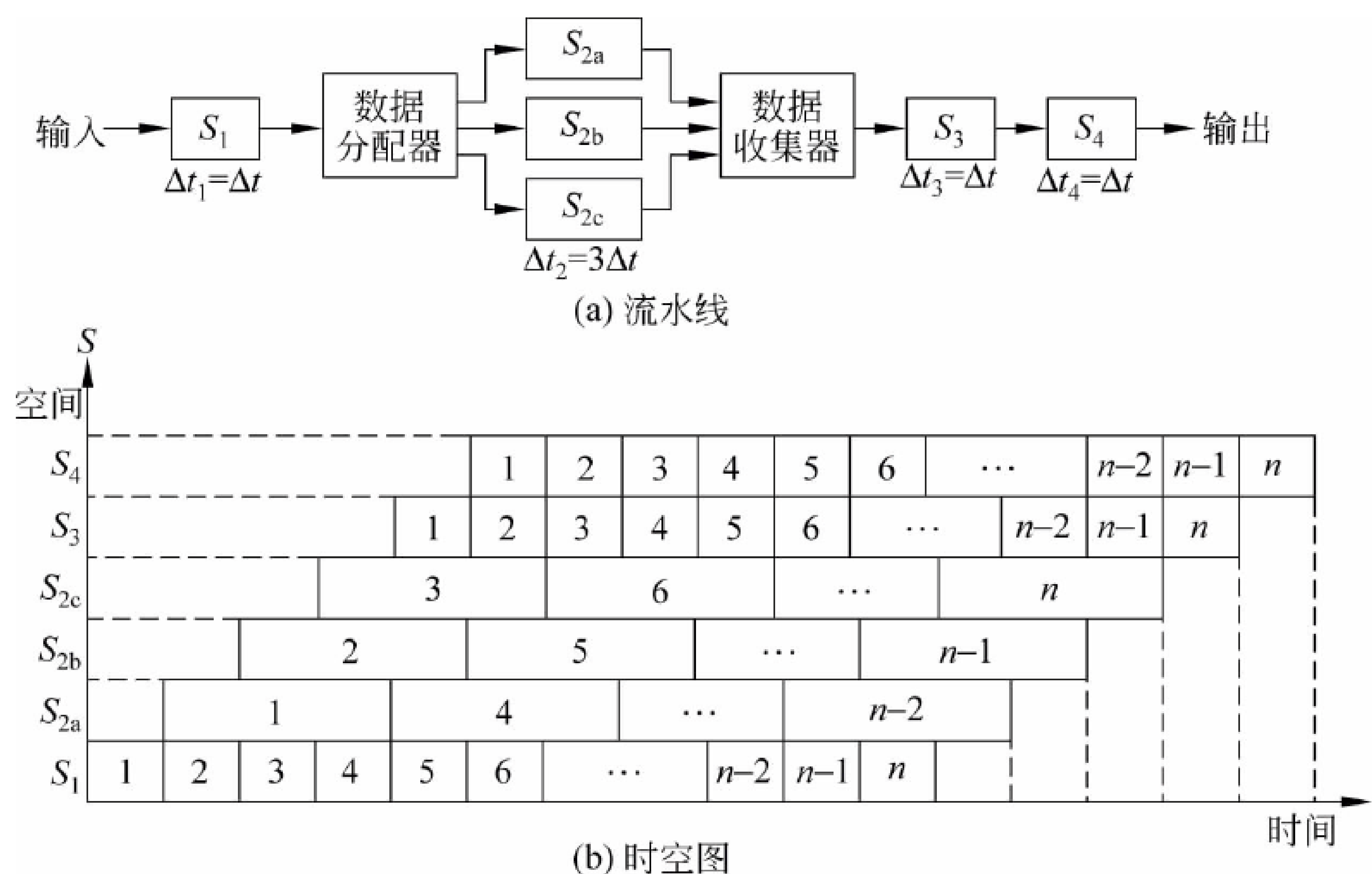


图 12.15　瓶颈流水段重复设置的流水线

采用瓶颈段资源重复设置的方法，其缺点是控制逻辑比较复杂。如图 12.15(a)所示，从流水段 S_1 到流水段 S_2 的各并列流水段之间需要设置一个数据分配器。它的任务是从 S_1 输出的第一个任务分配给 S_{2a}，从 S_1 输出的第二个任务分配给 S_{2b}，从 S_1 输出的第 3 个任务分配给 S_{2c}。之后依次重复。同样，并列流水段 S_{2a}、S_{2b}、S_{2c} 到流水段 S_3 之间需要设置一个数据收集器，分时收集前者的数据并输入到流水段 S_3 中。

12.2.2 流水线的加速比

完成同样一批任务,不使用流水线所用的时间与使用流水线所用的时间之比称为流水线的加速比(Speedup Ratio)。

设 T_0 表示不使用流水线,即顺序执行所用的时间,T_k 表示使用流水线时的执行时间,则流水线加速比 S 的基本公式为

$$S = \frac{T_0}{T_k} \tag{12.13}$$

如果流水线各段执行时间都相等,则一条 k 段流水线完成 n 个连续任务所需的时间为 $T_k=(k+n-1)\Delta t$。而不使用流水线,即顺序执行 n 个任务时,则所需的时间为 $T_0=k \cdot n \cdot \Delta t$,将 T_0 和 T_k 值代入式(12.13),得实际加速比为

$$S = \frac{kn\Delta t}{(k+n-1)\Delta t} = \frac{kn}{k+n-1} \tag{12.14}$$

上述情况下的最大加速比为

$$S_{\max} = \lim_{n \to \infty} \frac{kn}{k+n-1} = k \tag{12.15}$$

从式(12.15)中看出,当 $n>>k$ 时,线性流水线的各段执行时间均相等的情况下,流水线的最大加速比等于流水线的段数。

12.2.3 流水线的效率

流水线的设备利用率称为流水线的效率(Efficiency)。在时空图上,流水线的效率定义为完成 n 个任务占用的时空区有效面积与 n 个任务所用的时间与 k 个流水段所围成的时空区总面积之比。因此,流水线的效率包含了时间和空间两个因素。

n 个任务占用的时空区有效面积就是顺序执行 n 个任务所使用的总的时间 T_0,而 n 个任务所用的时间与 k 个流水段所围成的时空区总面积为 kT_k,其中 T_k 是流水线完成 n 个任务所使用的总时间,计算流水线效率的一般公式可以表示为

$$E = \frac{n\text{ 个任务占用的时空区有效面积}}{n\text{ 个任务所用的时间与 }k\text{ 个流水段所围的时空区总面积}} = \frac{T_0}{kT_k} \tag{12.16}$$

如果流水线的各段执行时间均相等,而且输入的 n 个任务是连续的,则一条 k 段流水线的效率为

$$E = \frac{kn\Delta t}{k(k+n-1)\Delta t} = \frac{n}{k+n-1} \tag{12.17}$$

式中,分子部分是完成 n 个任务实际占用的有效面积,分母部分是完成 n 个任务所用的时间与 k 个流水段所围成的总面积。因此,通过时空图来计算流水线的效率非常方便。

在流水线的各段执行时间均相等,输入到流水线的任务是连续的情况下,流水线的最高效率为

$$E_{\max} = \lim_{n \to \infty} \frac{n}{k+n-1} = 1 \tag{12.18}$$

显然,当 $n>>k$ 时,流水线的效率达到最大值1。这时流水线的各段均处于忙碌状态。从时空图中也可以看出,当 $n\to\infty$ 时分子和分母两部分的时空区面积接近于相等。

根据式(12.17)和式(12.7)，也可以得到式(12.19)，即当时钟周期不变时，流水线的效率与吞吐率成正比，即

$$E = \mathrm{TP}\Delta t \tag{12.19}$$

根据式(12.17)和式(12.14)，可以得出式(12.20)，此式表示流水线的效率 E 是流水线实际加速比 S 与它的最大加速比 k 之比。只有当效率 E 达到最大值即 $E=1$ 时，才能使实际加速比达到最大，即 $S=k$。

$$E = \frac{S}{k} \tag{12.20}$$

如果流水线的各段执行时间不相等，则除瓶颈流水段外，其他各流水段都有空闲时间，这些流水段的效率没有得到充分发挥，因此整个流水线的效率 E 比较低。

12.2.4 流水线的最佳段数

从上节分析看到，增加流水线段数 k 时，流水线的吞吐率和加速比都能提高。但是每一流水段输出端必须设置一个锁存器，当流水段数增多时，锁存器的总延迟时间也将增加，甚至有可能出现锁存器的总延迟时间超过流水线本身的延迟时间，并且流水线的价格也会增加。为此，要综合考虑各方面的因素，根据总体性能价格比来选择流水线最佳段数。

目前，一般处理机中的流水线段数在3～12之间，极少有超过15段的流水线。一般把8段或超过8段的流水线称为超流水线，采用8段以上流水线的处理机有时也称为超流水线处理机。

12.3 DLX指令集与DLX流水线

为了说明流水线的实现原理和讨论流水线中的相关问题，需要用到一种称为DLX的指令集结构，本节首先对DLX指令集结构进行简要介绍，并给出DLX的一种简单的实现。

DLX(读作Deluxe)是一种Load－Store型指令集结构。所谓Load－Store型又称为寄存器－寄存器型，是指对通用寄存器型指令集而言，其所有的ALU指令都不包含存储器操作数。与大多数当今的计算机类似，DLX强调：简单的Load－Store指令集；设计上重视流水线效率，包括固定长度指令编码；使编译器更容易产生高效的目标代码。

12.3.1 DLX指令集结构介绍

1. DLX的寄存器

DLX使用32个32位的通用寄存器(GPR)，分别命名为R0，R1，R2，…，R31。另外还有一组浮点寄存器(FPR)，这些浮点寄存器既可以作为32个32位的单精度寄存器，命名为F0，F1，F2，…，F31，也可以由相邻的偶数号和奇数号寄存器配对构成双精度寄存器，这种组合的双精度浮点寄存器在DLX中被命名为F0，F2，…，F28，F30。

寄存器R0的值总是0，后面会介绍利用该寄存器实现一些有用的操作，还有一些特殊的寄存器实现特定的功能。

2. DLX的数据类型

DLX提供了多种长度的整型数据和浮点数据，包括8位字节、16位半字、32位整数字，

以及32位单精度和64位双精度浮点数。

DLX的操作是面向32位的整数以及32位或者64位的浮点数的。字节或者半字在调入到32位寄存器时,用0或者符号位来填充32位寄存器的剩余部分。一旦载入,就将其作为32位整型数据进行处理。

3. DLX的寻址方式和数据传输

DLX可以提供寄存器寻址、立即数寻址、偏移寻址和寄存器间接寻址4种寻址方式,寄存器寻址字段的大小是5位,用于标识32个通用寄存器或者浮点寄存器。

DLX的内存按字节寻址,地址宽度为32位,而且用32位地址的高位字节在先的格式寻址。DLX是一种Load－Store结构,它通过寄存器(GPR和FPR)和存储器之间的数据传送完成对存储器的访问。

由于DLX支持上述的所有数据类型,所以与GPR有关的内存存取可以是一个字节、一个半字或者一个字,FPR可以加载或者存储单精度字或者双精度字。

4. DLX的指令格式

由于DLX的寻址方式很少,DLX的寻址方式被编码到指令操作码中。为了简化指令译码以及更容易进行流水线操作,所有的指令都是32位字长的,其中6位用来表示操作码。图12.16给出了DLX指令的I、R和J 3种类型的指令格式,所有的指令都依照这3种类型来编码。

6	5	5	16
操作码op	源寄存器1 rs_1	目的寄存器rd	立即数

字节、半字、字的加载和存储;
rd←rs_1 op立即数。

(a) I型指令

6	5	5	5	11
操作码op	源寄存器1 rs_1	源寄存器2 rs_2	目的寄存器rd	函数

寄存器－寄存器ALU操作:rd←rs_1 Func rs_2;
函数对数据的操作进行编码:加、减…;
对特殊寄存器的读/写和移动。

(b) R型指令

6	26
操作码op	与PC相加的偏移量

跳转,跳转并链接,陷阱和意外返回。

(c) J型指令

图12.16　DLX的指令格式

5. DLX的操作介绍

DLX除了支持上面提到的一些简单操作,还有一些其他操作。DLX指令大致可以分为4大类,即Load－Store操作、ALU操作、分支和跳转操作以及浮点数操作。为了对这些操作类型的论述方便,这里先对指令含义的表示方法约定如下。

(1) 符号"←"表示数据传送操作,当传送的数据位数不确切时,后面附带一个下标n,例如,"$\leftarrow_n$"表示传送一个n位数据。

(2) 上标表示复制一个域,例如,通过"0^{24}"可以得到一个24位全是0的域。

(3) 域的下标用来表示从域中选择某一位。域中的位从最高位开始标记,起始标记为0。下标可以表示一个单独的数字,如"$Regs[R4]_0$"表示寄存器R4中的符号位;下标也可以

表示一个范围，如"$Regs[R3]_{24..31}$"表示寄存器 R3 中的最低一个字节。

（4）符号"##"表示链接两个域，它可以出现在数据传送的任何一方。

（5）变量"Mem"用来表示主存，按照字节编址，可以传输任意字节的数据。

为了便于理解上面的表示方法，举例如下，假如 R8 和 R10 是 32 位的寄存器，那么 $Regs[R10]_{16..31} \leftarrow_{16} (Mem[Regs[R8]_0])^8$ ##（Mem[Regs[R8]]）表示的意思就是以 R8 的内容作为地址访问内存，得到的字节按符号位扩展到 16 位以后存入 R10 的低半字，而 R10 的高半字不变。下面就分别就 DLX 的 4 种类型的操作进行论述。

Load—Store 操作可以应用在所有的通用寄存器和浮点寄存器上，分别进行载入和存储操作。但需要注意的是对通用寄存器 R0 的 Load 操作没有任何效果，前面已经介绍过了，R0 的内容恒为 0。表 12.1 给出了 Load—Store 操作的一组实例。

表 12.1　Load—Store 指令实例

指令实例	指令名称	指 令 含 义
LW R1,30(R2)	加载整型字	$Regs[R1] \leftarrow_{32} Mem[30+Regs[R2]]$
LW R1,1000(R0)	加载整型字	$Regs[R1] \leftarrow_{32} Mem[1000+0]$
LB R1,40(R3)	加载字节	$Regs[R1] \leftarrow_{32} (Mem[40+Regs[R3]]_0)^{24}$ ## Mem[40+Regs[R3]]
LBU R1,40(R3)	加载无符号字节	$Regs[R1] \leftarrow_{32} 0^{24}$ ## Mem[40+Regs[R3]]
LH R1,40(R3)	加载整型半字	$Regs[R1] \leftarrow_{32} (Mem[40+Regs[R3]]_0)^{16}$ ## Mem[40+Regs[R3]] ## Mem[41+Regs[R3]]
LF F0,50(R3)	加载单精度浮点	$Regs[F0] \leftarrow_{32} Mem[50+Regs[R3]]$
LD F0,50(R2)	加载双精度浮点	Regs[F0] ## $Regs[F1] \leftarrow_{64} Mem[50+Regs[R2]]$
SW 500(R4),R3	存储整型字	$Mem[500+Regs[R4]] \leftarrow_{32} Regs[R3]$
SF 40(R3),F0	存储单精度浮点	$Mem[40+Regs[R3]] \leftarrow_{32} Regs[F0]$
SD 40(R3),F0	存储双精度浮点	$Mem[40+Regs[R3]] \leftarrow_{32} Regs[F0] Mem[44+Regs[R3]] \leftarrow_{32} Regs[F1]$
SH 502(R2),R31	存储整型半字	$Mem[502+Regs[R2]] \leftarrow_{16} Regs[R31]_{16..31}$
SB 41(R3),R2	存储整型字节	$Mem[41+Regs[R3]] \leftarrow_{8} Regs[R2]_{24..31}$

在 DLX 指令集结构中，所有的 ALU 指令都是寄存器—寄存器指令，包含了简单的算术运算和逻辑运算，如加、减、与、或、异或和移位。所有这些指令都支持立即数寻址模式，它带有一个 16 位的符号扩展立即数。LHI（加载高位立即数）操作把立即数加载到寄存器的高半部，而把低半部设为 0。这样可以通过两次 Load 指令构造一个 32 位的常数。

在 ALU 操作中，R0 经常被用来合成通用操作。加载一个常数的操作可以由一个立即数和一个源操作数是 R0 的加法来实现。寄存器—寄存器传送可以通过其中一个源操作数是 R0 的加法来完成。通常用 LHI 表示前者，而用 MOV 表示后者。

ALU 操作还有比较两个寄存器的比较指令（=、≠、<、>、≤、≥）。如果条件为真，则比较指令将在寄存器中放入一个 1，否则就放入一个 0。由于这些操作都设置寄存器，因此它们也被称为 Set-Equal，Set-Not-Equal，Set-Less-Than 等。同时这些比较指令也具有立即

数形式。表12.2给出了一组算术和逻辑指令的例子。

表12.2　ALU指令实例

指令实例	指令名称	指 令 含 义
ADD R1,R2,R3	加	Regs[R1]←Regs[R2]+Regs[R3]
ADDI R1,R2,#3	加立即数	Regs[R1]←Regs[R2]+3
LHI R1,#42	加载立即数到高半字	Regs[R1]←42##0^{16}
SLLI R1,R2,#5	逻辑左移立即数	Regs[R1]←Regs[R2]<<5
SLT R1,R2,R3	设置小于	If (Regs[R2]<Regs[R3]) Regs[R1]←1Else Regs[R1]←0

在DLX中,控制由一组跳转操作和一组分支操作来处理。4种跳转指令由指定目的地址的两种方式以及是否进行链接来区分。其中两种跳转指令用把26位带符号的偏移量加到程序计数器中的方法来确定目的地址;另外两种跳转指令通过指定包含目的地址的寄存器来确定。跳转有两种类型,一种是简单跳转,另一种是跳转并链接(用于过程调用)。后者将返回一个地址,即下一条顺序指令地址(返回地址),保存在R31中。

所有的分支指令都是条件分支指令。分支条件由指令确定,可能是测试源操作数寄存器是否为0;源操作数寄存器中可能有一个数值或者某个比较结果。分支的目标地址由26位带符号偏移量和程序计数器相加的结果决定。表12.3给出了一组典型的跳转和分支指令示例。

表12.3　典型的跳转和分支指令

指令实例	指令名称	指 令 含 义
J name	跳转	PC←name;$((PC+4)-2^{25})\leq name\leq((PC+4)+2^{25})$
JAL name	跳转并链接	Regs[R31]←PC+4;PC←name;$((PC+4)-2^{25})\leq name\leq((PC+4)+2^{25})$
JALR R2	寄存器型跳转并链接	Regs[R31]←PC+4;PC←Regs[R2]
JR R3	寄存器型跳转	PC←Regs[R3]
BEQZ R4,name	等于0时分支	If(Regs[R4]==0) PC←name;$((PC+4)-2^{25})\leq name\leq((PC+4)+2^{25})$
BNEZ R4,name	不等于0时分支	If(Regs[R4]!=0) PC←name;$((PC+4)-2^{25})\leq name\leq((PC+4)+2^{25})$

浮点操作是对浮点寄存器进行操作的指令。浮点指令同时还指明了相应的操作是单精度浮点操作还是双精度浮点操作。浮点操作包括加、减、乘、除。后缀F表示单精度浮点数,后缀D表示双精度浮点数(例如ADDD、ADDF、SUBD、SUBF、MULTD、MULTF、DIVD、DIVF)。浮点数比较指令会根据比较结果设置浮点数状态寄存器的某一位为1还是0,可以用两条分支指令BFPT和BFPF测试状态寄存器来决定是否进行分支。

浮点指令中,MOVF和MOVD分别把一个单精度或者一个双精度浮点寄存器的值,复制到另一个同类型的寄存器中。MOVFP2I和MOVI2FP在单精度浮点寄存器和整数寄存器之间传送数据。还有其他一些浮点指令请参考后续的指令列表。

表12.4给出了DLX的所有指令和其含义的列表。

表 12.4　所有的 DLX 指令列表

指令类型	指令操作码	指 令 含 义
数据传输	LB,LBU,SB	加载字节,加载无符号字节,存储字节
	LH,LHU,SH	加载半字,加载无符号半字,存储半字
	LW,SW	加载字,存储字
	LF,LD,SF,SD	加载单精度浮点数,加载双精度浮点数,存储单精度浮点数,存储双精度浮点数
	MOVI2S,MOVS2I	在通用寄存器和特殊寄存器之间传送数据
	MOVF,MOVD	将一个单精度/双精度浮点寄存器的内容复制到另一个单精度/双精度浮点寄存器中
	MOVFP2I,MOVI2FP	在整数寄存器和浮点寄存器之间传送 32 位的数据
算术/逻辑	ADD,ADDI,ADDU,ADDUI	带符号加,带符号立即数加,无符号加,无符号立即数加
	SUB,SUBI,SUBU,SUBUI	带符号减,带符号立即数减,无符号减,无符号立即数减
	MULT,MULTU,DIV,DIVU	带符号乘,无符号乘,带符号除,无符号除
	AND,ANDI	与,和立即数与
	OR,ORI,XOR,XORI	或,和立即数或,异或,和立即数异或
	LHI	加载立即数到高半字
	SLL,SRL,SRA,SLLI,SRLI,SRAI	移位:立即数形式和变量形式,包括逻辑左移,逻辑右移和算术右移
	S—,S—I	设置条件,"—"可以是 LT,GT,LE,GE,EQ,NE
控制	BEQZ,BNEZ	根据指定通用寄存器的内容等于还是不等于 0 分支
	BFPT,BFPF	测试浮点状态寄存器中的比较位为真还是假分支
	J,JR	跳转,基于寄存器的跳转
	JAL,JALR	跳转并链接,基于寄存器的跳转并链接
	TRAP	转到操作系统
	RFE	从异常恢复到用户模式
浮点操作	ADDD,ADDF	双精度浮点加,单精度浮点加
	SUBD,SUBF	双精度浮点减,单精度浮点减
	MULTD,MULTF	双精度浮点乘,单精度浮点乘
	DIVD,DIVF	双精度浮点除,单精度浮点除
	CVTF2D, CVTF2I, CVTD2F, CVTD2I,CVTI2F,CVTI2D	转换指令,CVTx2y 表示从类型 x 转换到类型 y,其中 x 和 y 可以是 I(整型)、D(双精度浮点)、F(单精度浮点)
	—D,—F	双精度浮点和单精度浮点比较,"—"可以是 LT,GT,LE,GE,EQ,NE,根据比较结果设置浮点状态寄存器中的位

12.3.2 DLX的一种简单实现

前面介绍了DLX这种Load—Store型指令集结构，这种指令集结构一个重要的特点就是便于实现指令流水，为了说明如何流水实现DLX的指令集结构，必须首先了解在非流水情况下DLX结构是如何实现的。以下部分就介绍一种简单的DLX实现方案，这种方案使引入流水的概念更加自然。

图12.17给出了实现DLX指令的一种简单的数据通路，从图中可以看出，每一条DLX指令的实现至多需要5个时钟周期。这5个时钟周期分别介绍如下。

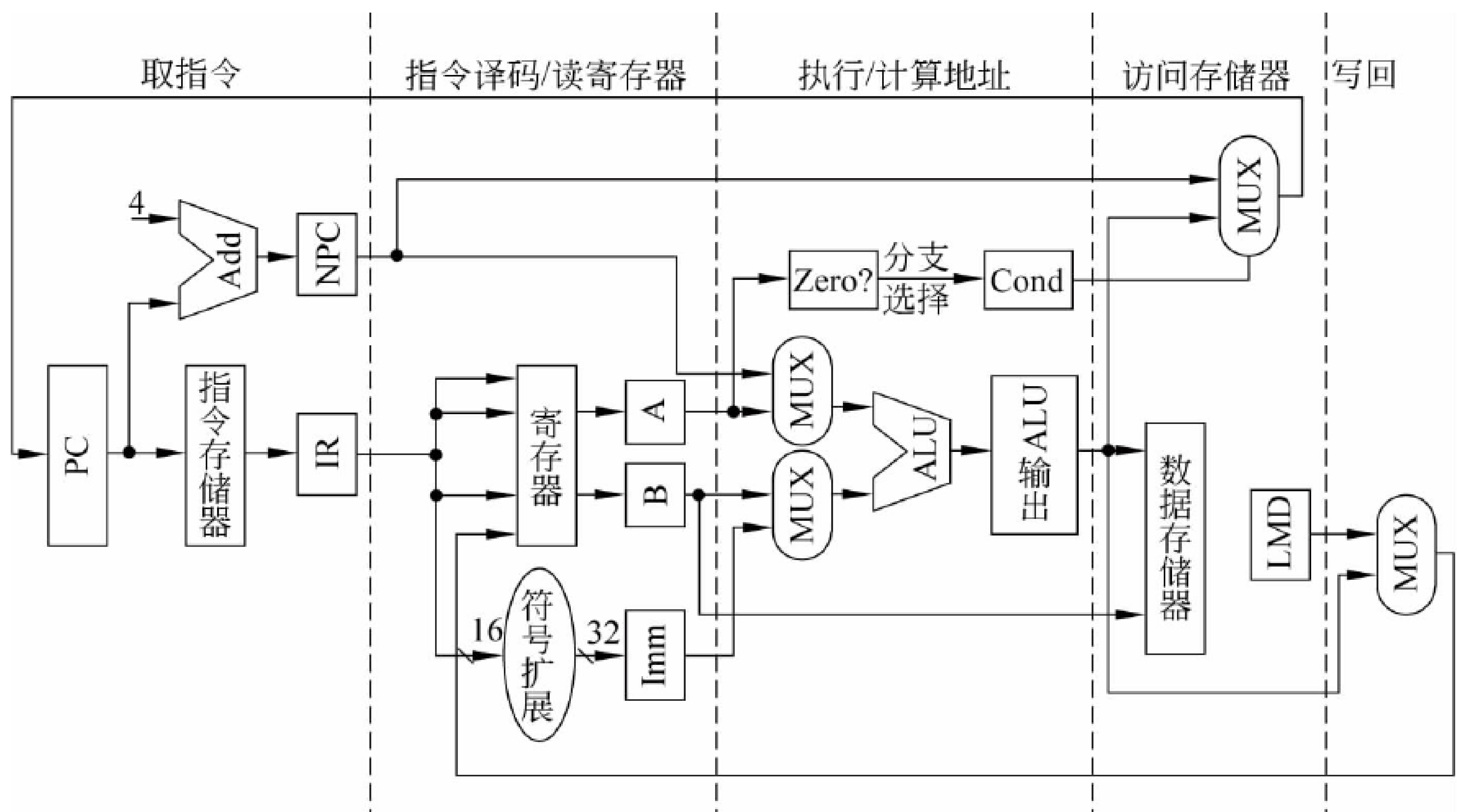


图12.17 DLX数据通路的实现

1. 取指令周期(IF)

这个周期的操作是根据程序计数器PC的值从存储器中取出指令，并将指令送入指令寄存器IR，同时PC的值加4，也就是指向顺序的下一条指令，并将下一条指令的地址放到寄存器NPC中。

$IR \leftarrow Mem[PC]$

$NPC \leftarrow PC+4$

2. 指令译码/读寄存器周期(ID)

这个周期的操作是进行指令译码，读指令寄存器IR，并将读出的结果放到两个临时寄存器A和B中。同时对IR寄存器内容的低16位进行符号扩展，然后将符号扩展后的32位立即数保存到临时寄存器Imm中。

$A \leftarrow Regs[IR_{6..10}]$

$B \leftarrow Regs[IR_{11..15}]$

$Imm \leftarrow ((IR_{16})^{16}\#\#IR_{16..31})$

3. 执行/有效地址计算周期(EX)

ALU对上一个周期准备好的操作数进行操作，根据DLX指令的不同类型执行下面4

个功能中的一个。

1）访问存储器

当指令为访问存储器指令时，此周期的操作是ALU将操作数相加形成有效地址，并将结果放到临时寄存器ALUoutput中。

ALUoutput←A+Imm

2）寄存器—寄存器ALU操作

当指令为寄存器—寄存器ALU操作指令时，此周期的操作是ALU根据操作码标示的功能对临时寄存器A和B中的值进行处理，并将结果放到临时寄存器ALUoutput中。

ALUoutput←A op B

3）寄存器—立即数ALU操作

当指令为寄存器—立即数ALU操作指令时，此周期的操作是ALU根据操作码标示的功能对临时寄存器A和Imm中的值进行处理，并将结果放到临时寄存器ALUoutput中。

ALUoutput←A op Imm

4）分支操作

当指令为分支指令时，此周期的操作是ALU将临时寄存器NPC和Imm中的值进行相加，得到分支的目标地址，将结果放到临时寄存器ALUoutput中。并对前一周期读到临时寄存器A中的值进行检查，决定分支是否成功。关系操作符op由分支操作码决定。

ALUoutput←NPC+Imm

Cond←(A op 0)

4. 存储器访问/分支完成周期(MEM)

在这个周期可能进行操作的DLX指令仅包括Load、Store和分支指令。下面就访问存储器操作和分支操作分别进行说明。

1）访问存储器操作

在需要访问存储器的操作里，如果是Load指令，将从存储器返回数据并放入LMD(Load Memory Data)寄存器中；如果是Store指令，就把寄存器B中的数据写入到存储器中。这两种情况下使用的地址在上一个周期中计算，并放到寄存器ALUoutput中。

LMD←Mem[ALUoutput]

或者

Mem[ALUoutput]←B

2）分支操作

如果进行分支操作，将被寄存器ALUoutput中的分支目标地址传送到程序计数器PC；否则将寄存器NPC中的内容传送到PC。

5. 写回周期(WB)

不同的指令在写回周期完成的任务也不一样，下面按照不同的指令类型对写回周期要完成的操作进行说明。

(1) 寄存器—寄存器型 ALU 指令：Regs[$IR_{16..20}$]←ALUoutput。

(2) 寄存器—立即数型 ALU 指令：Regs[$IR_{11..15}$]←ALUoutput。

(3) Load 指令：Regs[$IR_{11..15}$]←LMD。

这几种类型的指令都是将结果写入寄存器中。无论结果是来自存储器系统(LMD 中的内容)，还是来自 ALU 的计算结果(ALUoutput 中的内容)，都由操作码决定将其送入目标寄存器相应的域中。

图 12.17 给出了一条指令是如何沿着数据通路流动的。在每一个时钟周期结束的时候，所有在该时钟周期计算得到并且要在后面的时钟周期(用于本条指令或者下一条指令)需要的数据会被写入到存储器、通用寄存器、PC 或者临时寄存器等存储部件。临时寄存器在不同的时钟周期之间为同一条指令保存数据，而其他的存储单元的状态是可见的，它们在相邻的指令之间保存数据。在这种实现方案中，分支指令需要 4 个时钟周期，而其他指令需要 5 个时钟周期。

12.3.3 DLX 流水线的实现原理

在上一节中介绍的 DLX 的多周期实现的基础之上，如果在每一个时钟周期启动一条新的指令，就可以使图 12.17 中的数据通路成功流水。而上一节中的每一个时钟周期就是流水线的一个流水段，即流水线的一个周期。下面采用流水线结构的另外一种典型表示方法(即流水线时空图表示的另一种形式)来描述此流水线中指令的执行模式。如图 12.18 所示，每一条指令经过 5 个时钟周期执行完成，而在每一个时钟周期内，硬件将启动一条新的指令并执行 5 条不同指令的某个部分。

指　令	时　钟								
	1	2	3	4	5	6	7	8	9
指令 i	IF	ID	EX	MEM	WB				
指令 $i+1$		IF	ID	EX	MEM	WB			
指令 $i+2$			IF	ID	EX	MEM	WB		
指令 $i+3$				IF	ID	EX	MEM	WB	
指令 $i+4$					IF	ID	EX	MEM	WB

图 12.18　简单的 DLX 基本流水线

不幸的是，实际的流水线远没有我们上面说的这么简单。下面我们将对因为流水而引发的问题进行讨论。为了使多条指令能够在流水线中重叠执行，就必须保证在指令重叠时不会存在任何流水线资源冲突问题，即要保证流水线的各段在同一个时钟周期内不会使用相同的数据通路资源。图 12.19 给出了流水线方式下简化的 DLX 数据通路。该图显示了不同数据通路的重叠，其中周期 5(CC_5)表示稳定状态。由于寄存器在 ID 段被读，在 WB 段被写，它就出现了两次。在包围每个流水段的线框中，如果实线在右侧，说明是读操作；如果实线在左侧，说明是写操作；其他部分用虚线。IM 表示指令存储器，DM 表示数据存储器，CC 表示时钟周期。

从图 12.19 中还可以看到，主要的功能部件都在不同的时钟周期内使用，因而多条指令

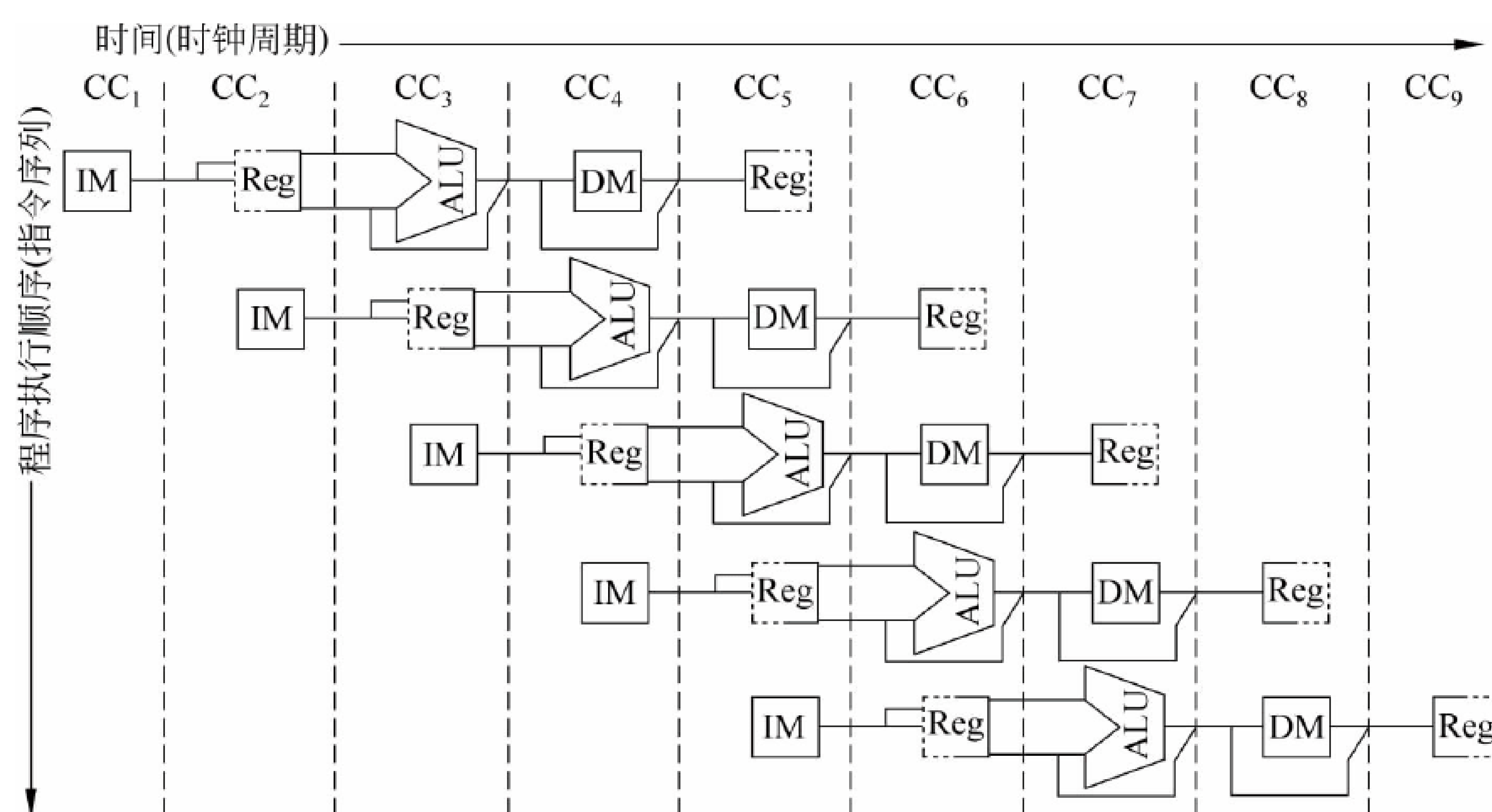


图 12.19　随时间移动的一系列数据通路

重叠执行时引入的冲突很少，这主要是因为以下 3 个方面的原因。

第一，前面介绍的基本数据通路中使用了分开的指令存储器(IM)和数据存储器(DM)，典型的实现方式是采用指令 Cache 和数据 Cache。这样设计的好处是避免了取指令操作和访问数据操作之间存在的存储器访问冲突。

第二，在两个流水线段都使用了寄存器：ID 段读，WB 段写。这两个使用是截然不同的，所以我们在两个地方画出了寄存器，那么如果读操作和写操作都是对同一个寄存器进行，如何解决这个问题我们后续将进行分析。

第三，图 12.19 中没有考虑 PC 的问题。为了在每一个时钟周期能够启动一条新的指令，就需要每个周期都对 PC 进行自加运算并写回，对于上述的 DLX 流水线，这项工作必须在 IF 段完成，以便为下一条指令做好准备。如果考虑到分支指令的影响，就出现新的问题，因为分支指令可能要改变 PC 的值，但是它仅在 MEM 段结束时才改写 PC 值。针对这个问题，就需要对上述流水线的数据通路重新进行组织，争取只在 IF 段完成改变 PC 值的操作。这种变化实际上是一个如何处理分支的问题，这个问题将在下节中进行分析。

前面已经讨论过，在 DLX 基本流水线中，每一个时钟周期都要用到所有的流水段，一个流水段中的所有操作必须在一个时钟周期内完成。特别是数据通路的流水要求从一个流水段到下一个流水段的数据必须要放到寄存器中。图 12.20 是对图 12.19 的数据通路进行改进后的流水线数据通路，在流水线的各个流水段之间加入了被称为流水线寄存器(流水线锁存器)的寄存器堆，并在这些寄存器堆上标明所连接的流水段。

所有用于在同一条指令的各个时钟周期之间保存临时数据的寄存器，都归入流水线寄存器这一类中。流水线寄存器由它们相连的流水段的名称来标记。例如，IF/ID 流水线寄存器就是连接 IF 流水段和 ID 流水段的，指令寄存器 IR 是 IF/ID 流水线寄存器的一个字段，记作 IF/ID. IR。

这些流水线寄存器保存着从一个流水段传送到下一个流水段的所有数据和控制信息。随着流水过程的进行，这些数据和控制信息从一个流水线寄存器复制到下一个流水线寄存器，直到不再需要为止。如果在流水过程中仅仅使用先前非流水线数据通路中的临时寄存

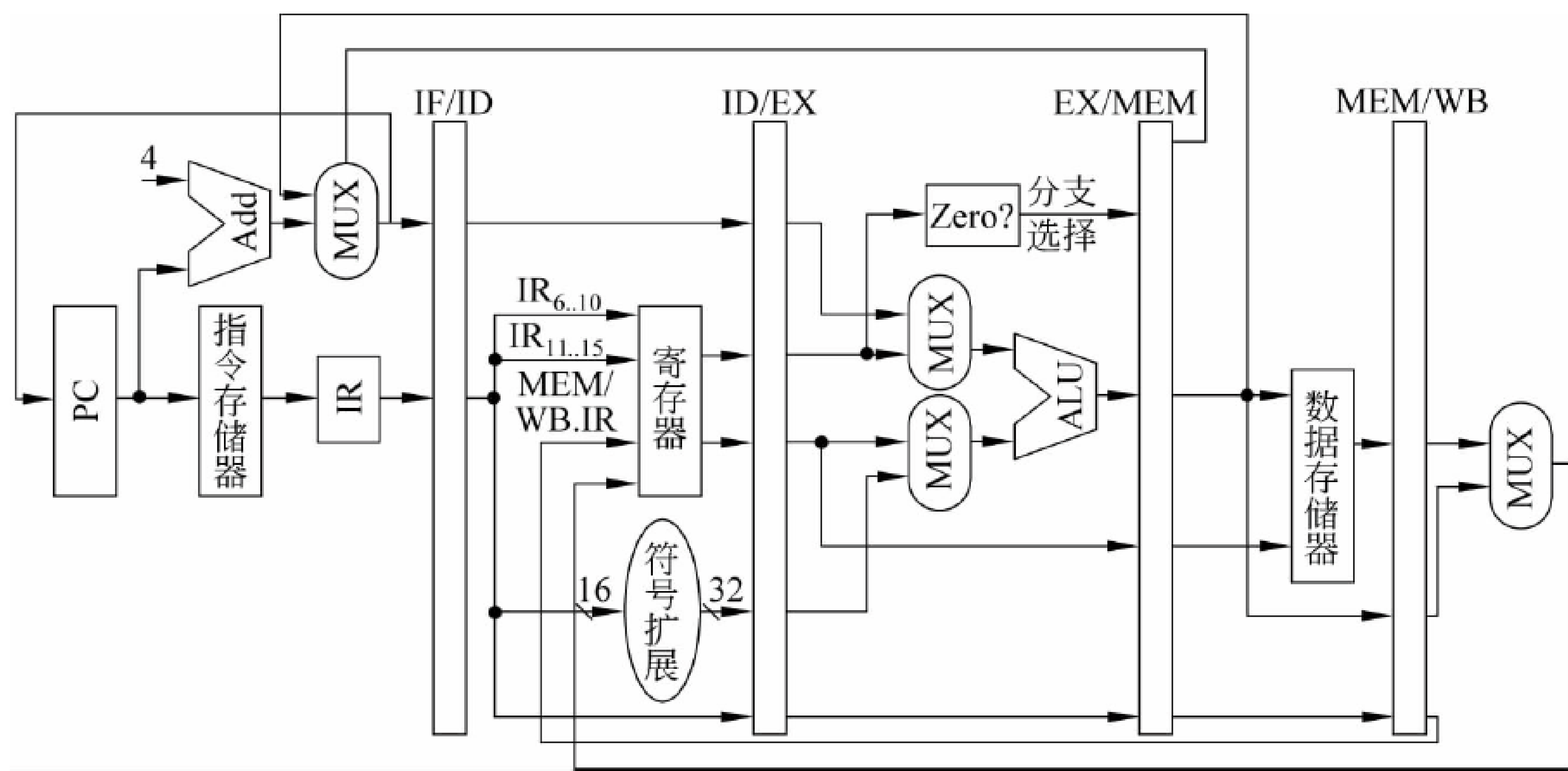


图 12.20 DLX 流水线的数据通路

器,那么当这些寄存器保存的临时值还在为流水线中某条指令所用时,就可能会被流水线中的其他指令所重写。

值得注意的是,图 12.17 中的 PC 值多路选择器在图 12.20 中已经被移到 IF 段,这样做的目的是保证对 PC 值的写操作只出现在一个流水段内,否则当分支转移成功的时候,流水线中两条指令都试图在不同的流水段修改 PC 值,从而发生写冲突。

每个时刻,每条指令都只在一个流水段上是活动的,因此,任何指令所作的任何动作都发生在一对流水线寄存器之间。所以,我们就可以根据指令的类型检查,在任一个流水段都要做些什么来看出流水线的动作。表 12.5 给出了 DLX 流水线的每个流水段的操作。

表 12.5 DLX 流水线的每个流水段的操作

流水段	任意指令		
IF	IF/ID. IR←Mem[PC]; IF/ID. NPC,PC←(if EX/MEM. cond {EX/MEM. ALUOutput} else {PC+4})		
ID	ID/EX. A←Regs[IF/ID. $IR_{6..10}$];ID/EX. B←Regs[IF/ID. $IR_{11..15}$]; ID/EX. NPC←IF/ID. NPC;ID/EX. IR←IF/ID. IR; ID/EX. Imm←(IF/ID. IR_{16})^{16} ## IF/ID. $IR_{16..31}$;		
	ALU 指令	Load/Store 指令	分支指令
EX	EX/MEM. IR←ID/EX. IR; EX/MEM. ALUOutput← ID/EX. A op ID/EX. Imm; EX/MEM. cond←0;	EX/MEM. IR←ID/EX. IR; EX/MEM. ALUOutput← ID/EX. A+ID/EX. Imm;	EX/MEM. ALUOutput← ID/EX. NPC+ID/EX. Imm; EX/MEM. cond←ID/EX. A op 0;
MEM	MEM/WB. IR← EX/MEM. IR; MEM/WB. ALUOutput← EX/MEM. ALUOutput;	MEM/WB. IR← EX/MEM. IR;MEM/WB. LMD← Mem[EX/MEM. ALUOutput]; 或 Mem[EX/MEM. ALUOutput]← MEM/WB. LMD	

续表

流水段	任 意 指 令		
WB	Regs[MEM/WB. $IR_{16..20}$]← MEM/WB. ALUOutput; 或 Regs[MEM/WB. $IR_{11..15}$]← MEM/WB. ALUOutput;	Regs[MEM/WB. $IR_{11..15}$]← MEM/WB. LMD;	

为了控制这个简单的 DLX 基本流水线，我们还需要确定如何控制图 12.20 中数据路径的 4 个多路开关。ALU 输入端的两个多路开关根据指令类型来设置，它由 ID/EX 寄存器的 IR 给出。其中，上面的 ALU 输入多路开关根据指令是否是分支指令来确定，下面的多路开关根据指令是寄存器一寄存器 ALU 指令还是其他类型的指令来确定。IF 段的多路开关选择增加以后的 PC 值或者 EX/MEM. NPC 的值(分支的目标地址)作为下一条指令的地址，这个多路开关由 EX/MEM. cond 来控制。第 4 个多路开关由 WB 段的指令是 Load 指令还是 ALU 指令来进行控制。

12.4　流水线中的相关问题

在流水线中经常有一些被称为“相关”的情况发生，它使得指令序列中下一条指令无法在设计的时钟周期执行，这些“相关”会降低流水线可以获得的理想性能。一般来说，流水线中的相关可以分为以下 3 种类型。

第一种是结构相关，它是指令在重叠执行的过程中，硬件资源满足不了指令重叠执行的要求，发生硬件资源冲突而产生的相关。

第二种是数据相关，它是指在同时重叠执行的几条指令中，一条指令依赖于前面指令的执行结果的数据，但是目前尚不能得到这个数据时发生的相关。

第三种是控制相关，它是指流水线中分支指令或者其他需要改写 PC 的指令造成的相关。

流水线相关问题是流水线执行过程中的主要障碍，会给流水线中指令序列的顺利执行带来许多不利的影响。如果不能较好地处理流水线相关问题，就可能影响流水线的性能，甚至使程序运行产生错误的结果。要消除相关带来的问题，就要求正常处理流水线中的部分指令，而延迟另一部分指令的执行，也就是暂停一些指令的执行过程。

12.4.1　结构相关

处理器进行流水时，如果希望在流水线中顺利执行指令的所有组合，势必要求流水线的功能部件能够充分流水，或者资源重复设置。因为资源冲突而无法使用某种指令的组合，就称该处理器的流水线产生了结构相关。最常见的结构相关出现在部分功能部件不能充分流水的时候，即使用此部件的多条指令不能按照每个时钟周期前进一拍的速率流水。另外一种常见的结构相关是因为部分资源没有充分重复设置，导致流水线中若干指令不能同时执行。

很多流水线机器的指令和数据都共享一个存储器，如果在某个时钟周期内，流水线既要

完成某条指令到存储器读取数据的操作,又要完成另一条指令到存储器的读取指令的操作,这样就会发生存储器访问冲突问题,即产生结构相关。如图12.21所示就是这种情况。

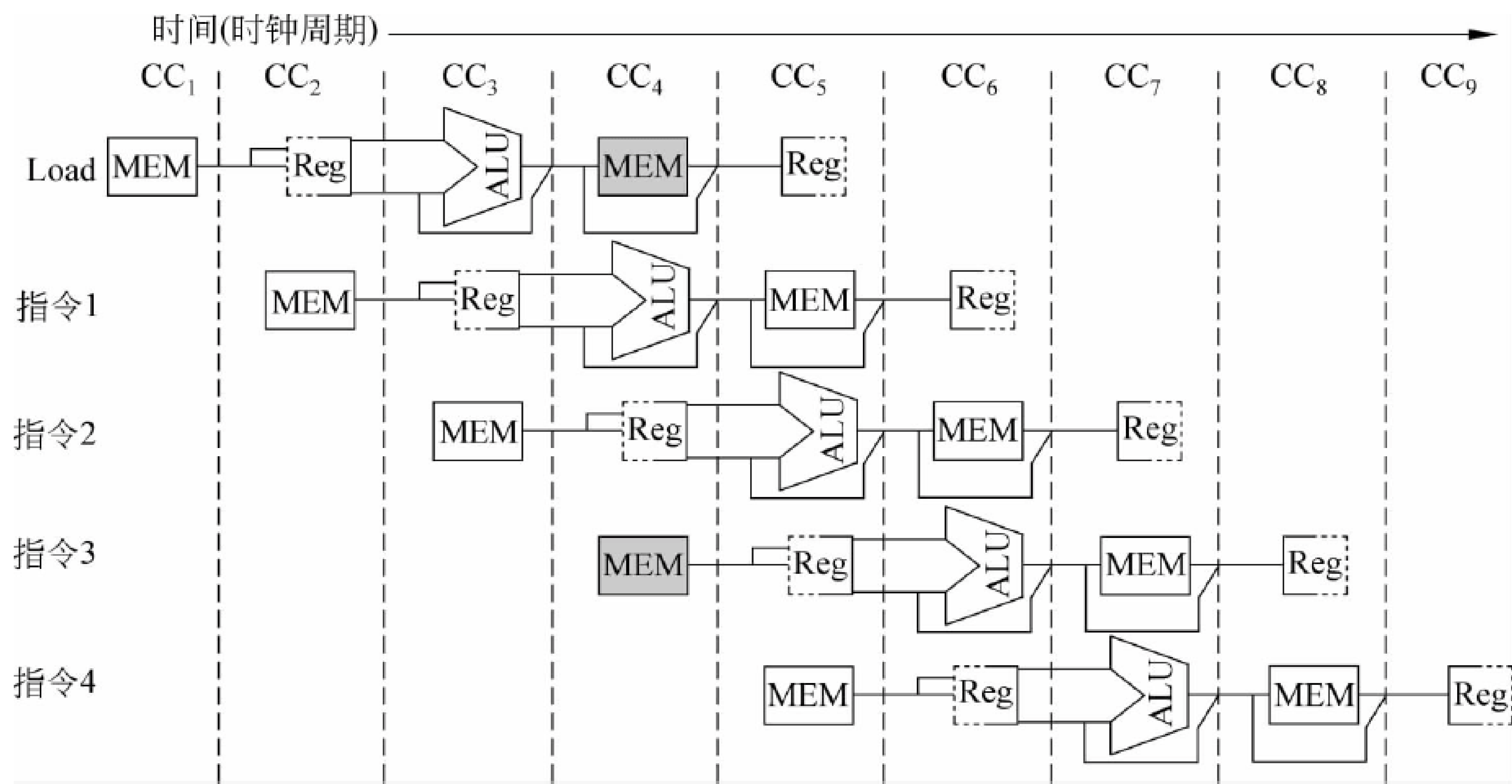


图12.21 存储器访问冲突造成的结构相关

为了消除这种冲突,当前一条指令需要访问数据存储器时,就暂停另一条指令的取指操作,也就是把流水线暂停一个时钟周期,这个时钟周期叫作流水线的一个暂停周期,或者称为"流水线气泡"(简称气泡)。图12.22表示加入一个暂停周期后的数据通路。

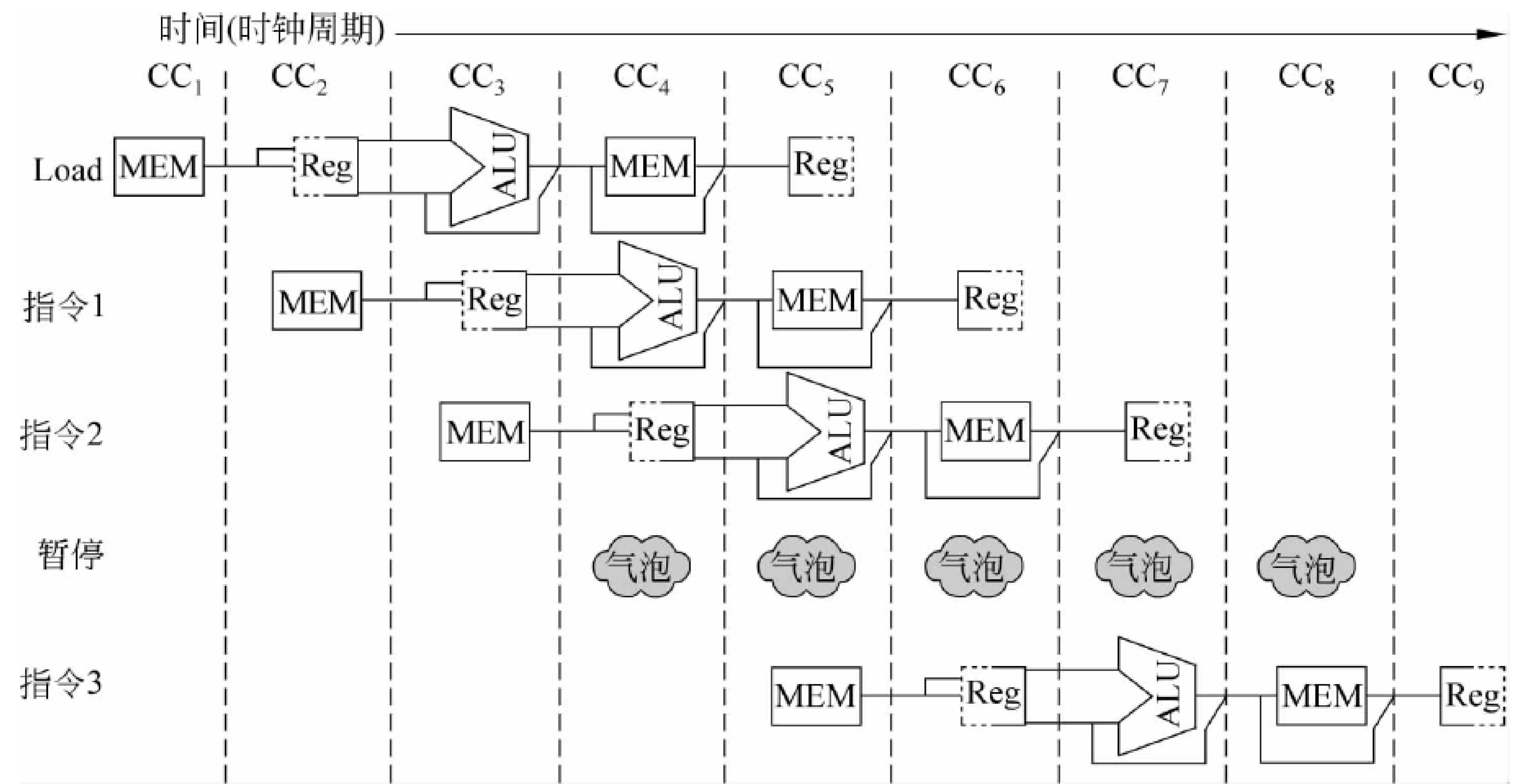


图12.22 插入流水线气泡消除流水线结构相关

可以使用图12.23所示的时空图来表示插入暂停的流水线。图中用Stall标记暂停周期,并将指令 $i+3$ 的取指令操作右移一个时钟周期,因此,要到第9个时钟周期流水线才能完成指令 $i+3$,而在第8个时钟周期没有任何指令流出。

为了消除结构相关引入了暂停周期,这必然要降低流水线的性能。因此,可以考虑采用资源充分重复设置的方法来避免结构相关。一种常用的方法就是在流水线机器中设置互相独立的指令存储器和数据存储器,具体而言就是将Cache分为指令Cache和数据Cache。

指令	时钟									
	1	2	3	4	5	6	7	8	9	10
指令 i	IF	ID	EX	MEM	WB					
指令 $i+1$		IF	ID	EX	MEM	WB				
指令 $i+2$			IF	ID	EX	MEM	WB			
指令 $i+3$				Stall	IF	ID	EX	MEM	WB	
指令 $i+4$						IF	ID	EX	MEM	WB
指令 $i+5$							IF	ID	EX	MEM
指令 $i+6$								IF	ID	EX

图 12.23 引入暂停的流水线时空图

在其他因素对流水线的性能影响都相同的情况下，没有结构相关的流水线机器的CPI较小，为什么设计时还允许存在结构相关呢？主要原因有两个，降低成本和减少功能部件的延迟。为了避免结构相关，确保流水线的各种功能部件完全流水化，或者重复设置足够的硬件资源，都会导致成本过高。例如，要求在流水线机器中，每个时钟周期内都能同时进行取指令操作和访问数据的存储器操作，而又不发生结构相关，存储总线的带宽就要加倍。类似的是，完整流水的浮点乘法器需要很多逻辑门。为了消除很少出现的结构相关而增加更多的硬件，成本就显得太高了。另外，完全可以设计出比完全流水化功能部件具有更短延迟时间的非流水和不完全流水化的功能部件，较短的延迟是因为不需要流水线寄存器。

12.4.2 数据相关

指令完全串行执行，不会产生数据相关问题，而在指令流水线技术中，同时处在流水线中的后续指令用到其前面指令的运算结果，而前面指令的运算结果尚未产生或者尚未送到指令位置时，就会导致数据相关。从如图 12.24 所示的例子中指令在流水线中的执行情况的分析，可以清楚地看到这种数据相关的情况。

在图 12.24 中，ADD 指令后的所有指令都用到了 ADD 指令的计算结果。ADD 指令在 WB 段才能将计算结果写入寄存器 R1，但是 SUB 指令在 ID 段就要读取 R1 中的值，这种情况下就产生了数据相关。除非采取预防措施，否则 SUB 指令读出的将是错误的数据，并且导致错误的计算结果。AND 指令也受这种数据相关的影响，从图 12.24 可以看出，直到时钟周期 5 写 R1 才能完成，AND 指令在时钟周期 4 取出的 R1 的值也是错误的。XOR 指令得到了正确执行，因为这条指令在时钟周期 6 读取寄存器 R1，而此时 R1 已经被正确地写入，该指令可以顺利执行。其实 OR 指令也可以得到了正确执行，这只需要使用一个已经在流水线图表中画出的简单技巧即可，就是在前半个周期写寄存器堆，后半个周期读寄存器堆。这种技巧通过在前面图中在寄存器周围画的虚线框来表示。

为了保证上面给出的指令序列的正确执行，流水线只好暂停 ADD 指令之后的所有指令，直到 ADD 指令将计算结果写入寄存器 R1 之后，再启动 ADD 指令之后的指令继续执行。那么有没有其他方法解决 SUB 指令和 AND 指令的数据相关呢？下面要介绍的定向

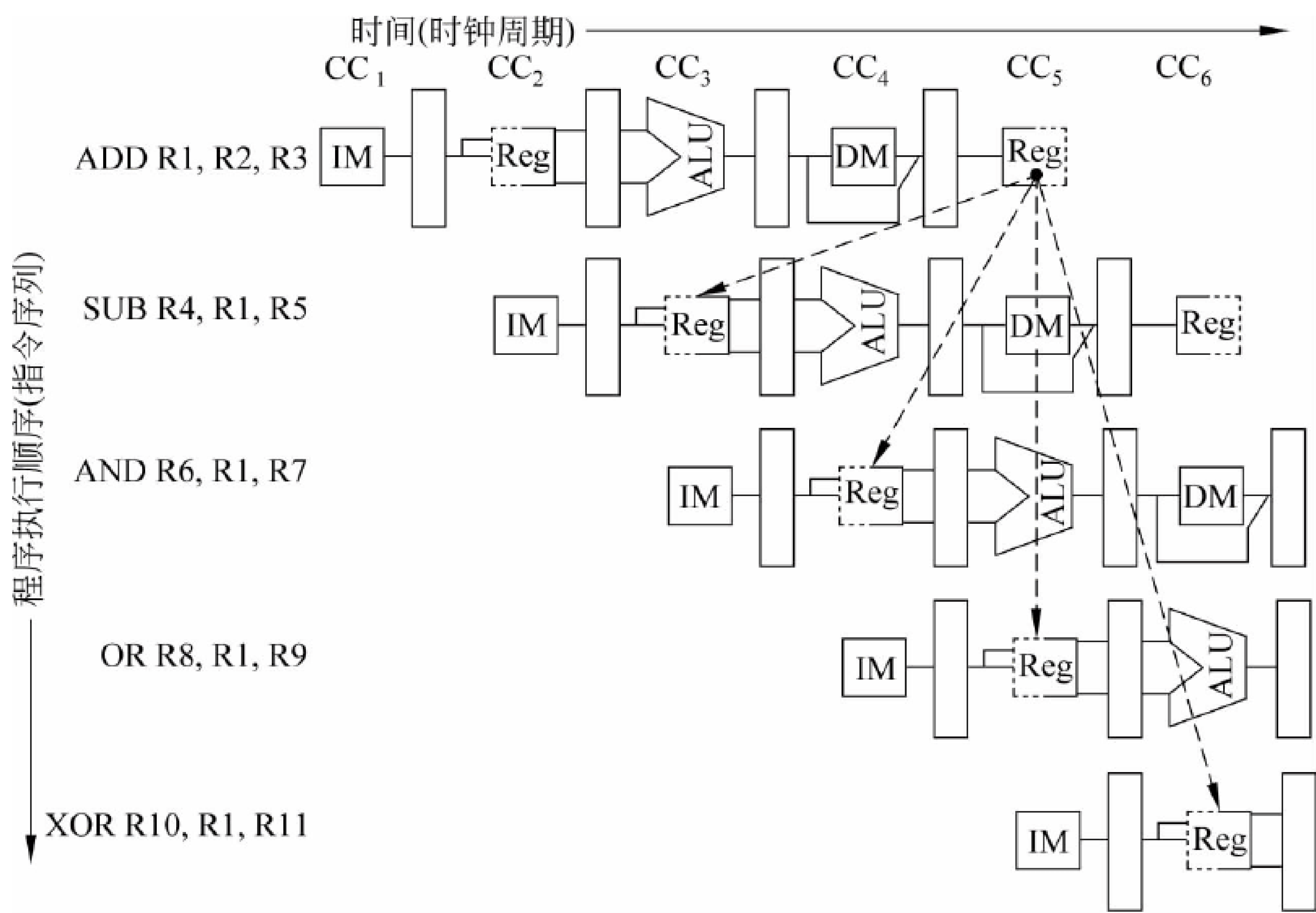


图 12.24 流水线数据相关示例

技术就可以减少数据相关带来的流水线暂停。

1. 减少数据相关带来暂停的定向技术

图 12.24 中的问题可以通过一种简单的技术来解决,这种技术被称为定向技术(也称为旁路技术)。定向技术的主要思想是:在某条指令产生一个计算结果之前,其他指令并不会真正需要使用这个计算结果,如果能够从这个计算结果产生的地方直接将它送到后续其他指令需要使用它的地方,那么就可以避免暂停。对前面的例子而言,SUB 指令是在 ADD 指令产生了计算结果之后,才真正使用这个结果的。如果把 ADD 指令的计算结果从 EX/MEM 寄存器之间送到 SUB 指令需要的地方,即 ALU 的输入锁存器,那么就可以不用引入暂停了。

定向技术的工作流程可以归纳如下。

(1) 从 EX/MEM 寄存器送入 ALU 的结果总是反馈到 ALU 的输入锁存器。

(2) 当定向硬件检测到前一个 ALU 运算结果的写入寄存器就是当前 ALU 操作的源寄存器时,那么控制逻辑就将前一个 ALU 运算结果定向到 ALU 的输入,而不是再从寄存器堆读出源寄存器内容。

从图 12.24 中可以看出还需要注意的是,需要定向的运算结果可能不止来自前一条指令的计算结果,还可能是前面与其不相邻的其他指令的计算结果。图 12.25 是采用了定向技术之后上述指令序列的执行情况,从图中可以看到采用定向技术之后这个指令序列可以顺利执行而不需要暂停,图中的虚线标明了定向路径。

定向技术的思想可以一般化,就是把运算结果直接送到需要它的功能部件,即一个部件的输出直接送到另一个部件的输入。考虑到 DLX 流水线的实现,就需要从任意一个流水线寄存器到任意一个功能部件输入端的定向。因为 ALU 和数据存储器都接收操作数,就需要从 EX/MEM 和 MEM/WB 寄存器到它们输入端的旁路。此外,DLX 在 EX 周期使用了一个检测零的部件,所以还需要到这个部件的定向。

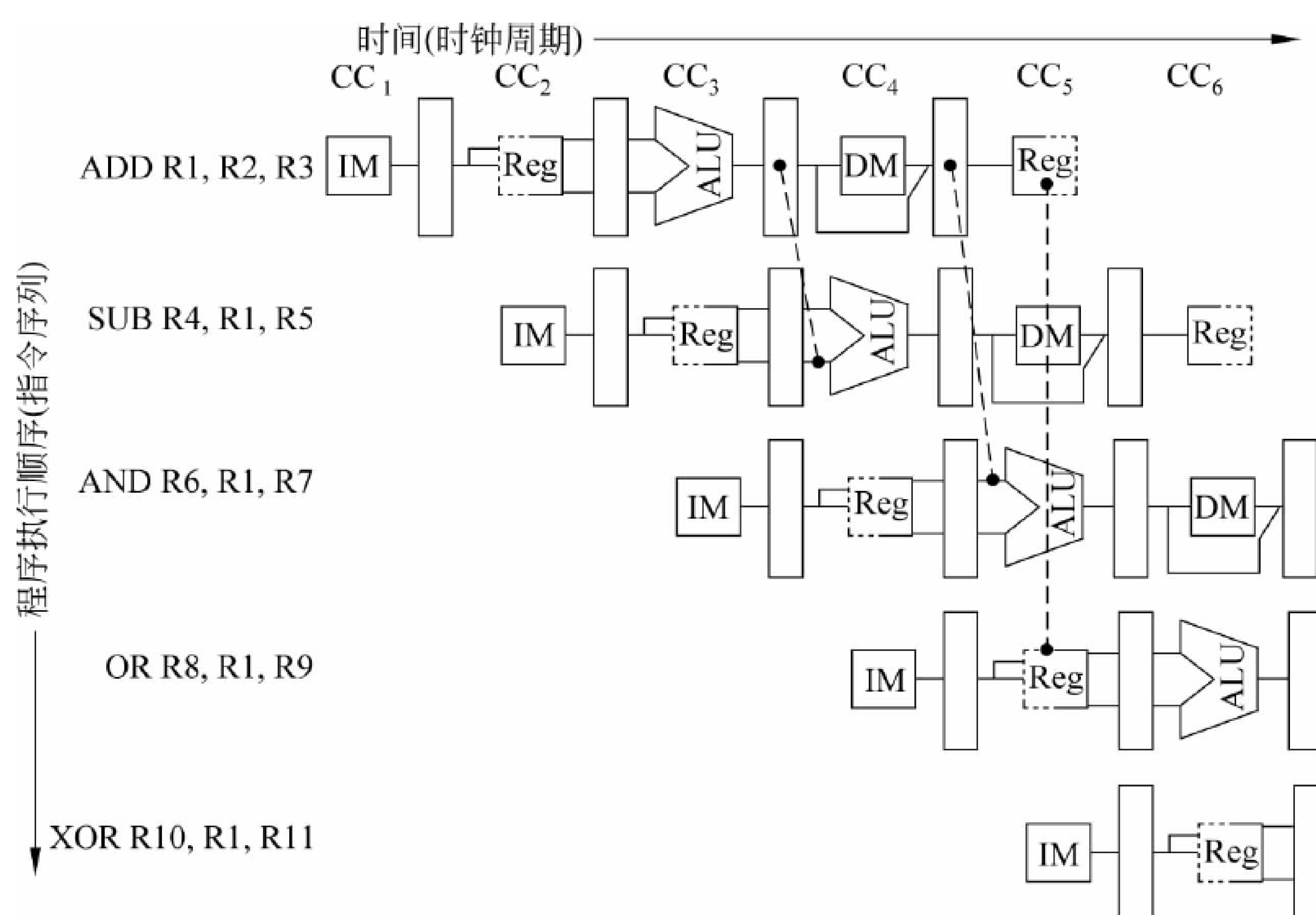


图 12.25 采用定向技术消除数据

图 12.26 改进了图 12.20 的设计，图中的虚线表示流水线所增设的定向路径。从中可以看到，要定向到 ALU，需要为每个 ALU 多路开关增加 3 个输入端，以及到这 3 个输入端的相应通路。这 3 条新通路分别来自 EX 段末尾的 ALU 输出端、MEM 段末尾的 ALU 输出端和 MEM 段的存储器输出端。

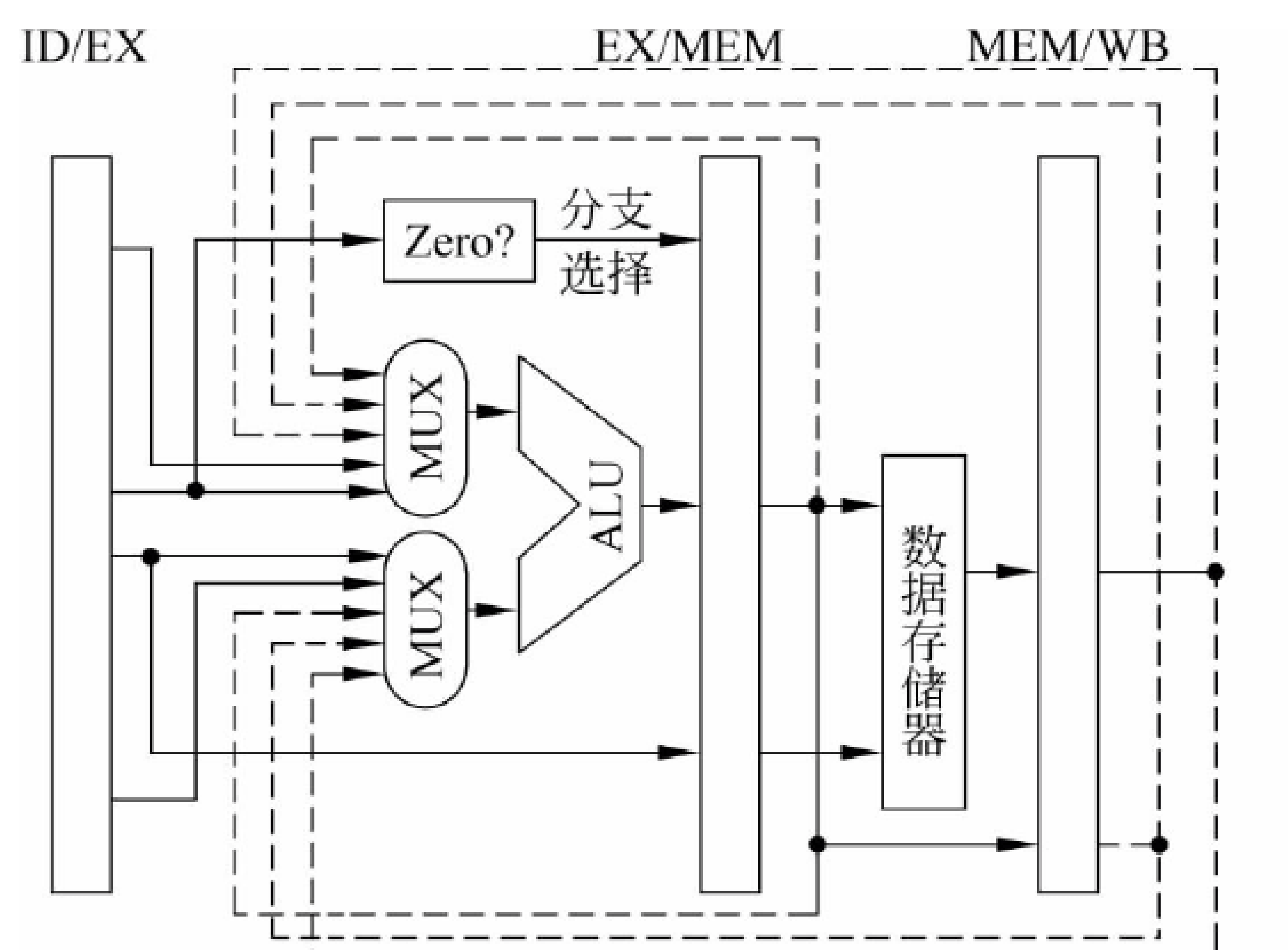


图 12.26 DLX 流水线增设的定向路径

2. 必须进行暂停的数据相关

虽然利用定向技术可以消除由于数据相关带来的暂停，但是不幸的是，并非所有的数据相关都能用定向技术来解决。请看图 12.27 中的指令序列，这个指令序列在流水线中执行时所需要的定向路径在图 12.20 中给出，它与带反馈的 ALU 操作的情形不同。LW 指令要到第 4 个时钟周期末才能从存储器中读出数据，而 SUB 指令在第 4 个时钟周期开始的时候就需要这一数据。因此，不能用简单的硬件来消除 Load 操作造成的数据相关。如图 12.27

所示，这种定向需要有一个在时间上反向流动的定向路径，这是计算机设计人员无法实现的。

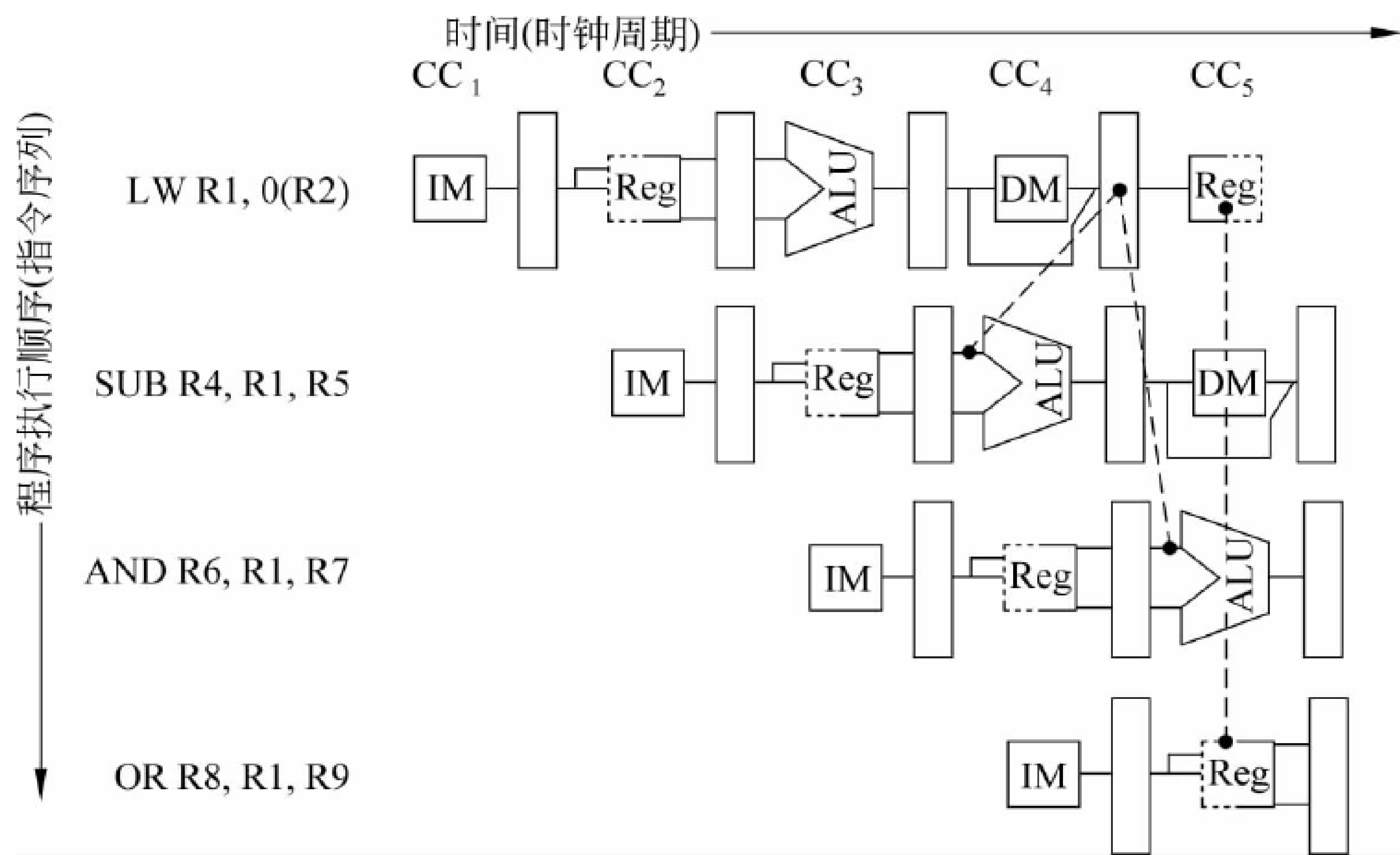


图 12.27　LW 指令不能将结果定向到 SUB 指令

为了保证流水线能够正确执行上述的指令序列，需要加入一种称为“流水线互锁”(Pipeline Interlock)的新的功能部件。通常，流水线互锁检测到上述的数据相关后就暂停流水线，直到能够通过定向技术解决数据相关为止。图 12.28 表示的是引入了暂停和合理的定向的流水线，由于暂停 SUB 以后的指令都后移了一个时钟周期，到 AND 指令就变成了通过寄存器堆来定向，而 OR 指令根本就不需要旁路了。

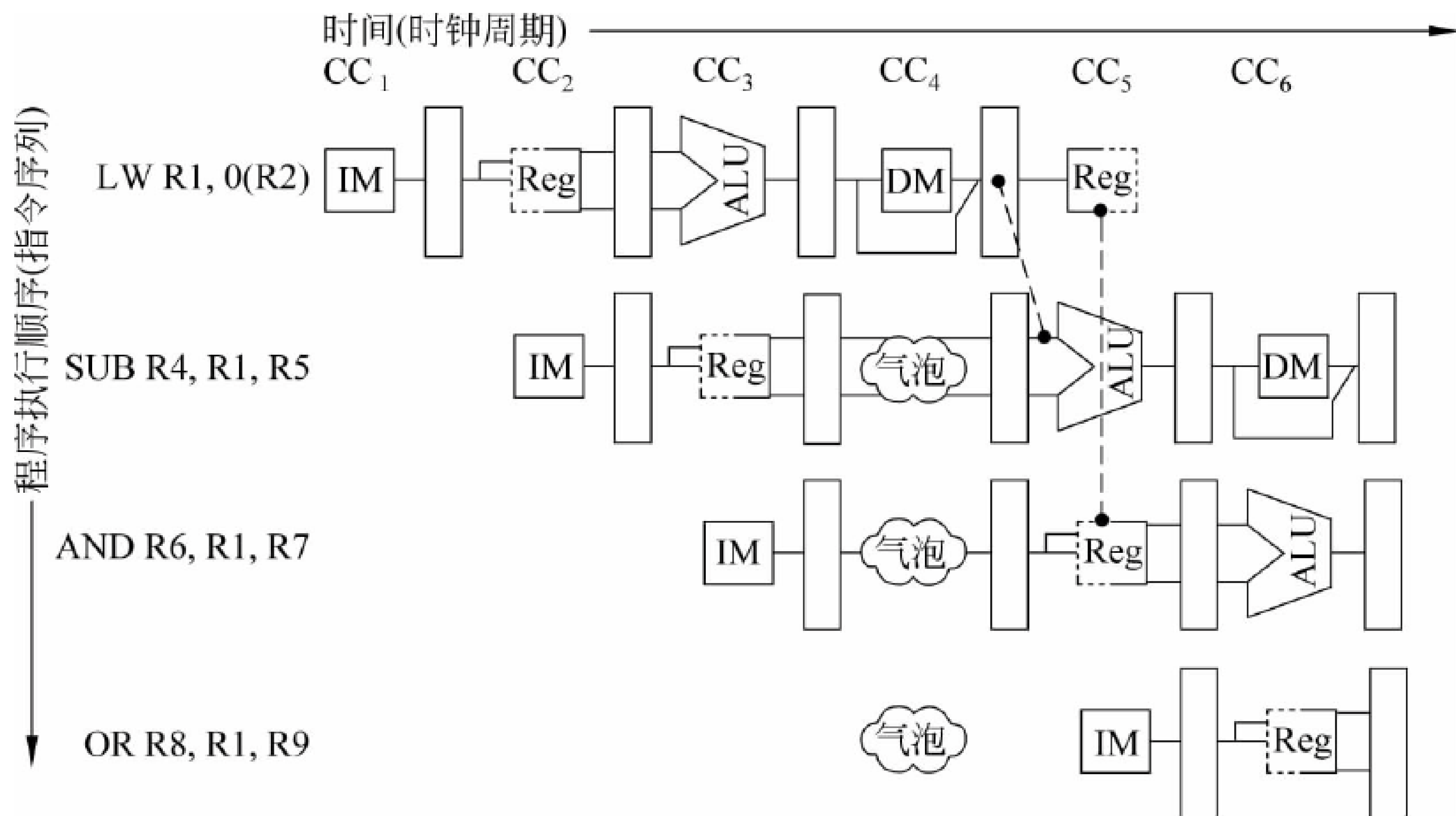


图 12.28　插入暂停后的流水线数据通路

暂停引入的流水气泡需要多用一个时钟周期来完成这个指令序列。在时钟周期 4 没有启动新的指令，在时钟周期 6 也没有流出任何执行完毕的指令。图 12.29 给出了加入暂停前后的流水线时空图。

指　令	时　钟							
	1	2	3	4	5	6	7	8
LW R1,0(R2)	IF	ID	EX	MEM	WB			
SUB R4,R1,R5		IF	ID	EX	MEM	WB		
ADD R6,R1,R7			IF	ID	EX	MEM	WB	
OR R8,R1,R9				IF	ID	EX	MEM	WB

(a) 加入暂停之前

指　令	时　钟								
	1	2	3	4	5	6	7	8	9
LW R1,0(R2)	IF	ID	EX	MEM	WB				
SUB R4,R1,R5		IF	ID	Stall	EX	MEM	WB		
ADD R6,R1,R7			IF	Stall	ID	EX	MEM	WB	
OR R8,R1,R9				Stall	IF	ID	EX	MEM	WB

(b) 加入暂停之后

图 12.29　加入暂停前后流水线时空图

3. 编译器调度方法处理数据相关

流水线不仅会遇到多种类型的暂停,而且在流水线中有很多种暂停会频繁出现。例如对于最常见的 $A=B+C$ 这样的操作形式,采用比较典型的代码生成方法可以得到图 12.30 中的指令序列,这个指令序列的流水线时空图如图 12.30 所示,从中可以看出,在 ADD 指令的流水过程中必须插入一个暂停时钟周期,以保证变量 C 的读入值有效;而 SW 指令不需要另外的暂停,因为加法的结果可以直接定向到数据存储器供 SW 指令使用。

指　令	时　钟								
	1	2	3	4	5	6	7	8	9
LW R1,B	IF	ID	EX	MEM	WB				
LW R2,C		IF	ID	EX	MEM	WB			
ADD R3,R1,R2			IF	ID	Stall	EX	MEM	WB	
SW A,R3				IF	Stall	ID	EX	MEM	WB

图 12.30　$A=B+C$ 的典型 DLX 指令序列流水线时空图

对于定向技术无法消除的数据相关问题还是有其他方法能够进行处理的,通过编译器对指令的调度,即重新排列指令序列的顺序就可以达到这个目的。这种重新组织代码顺序的技术一般被称为"流水线调度"或者"指令调度"。

下面我们举一个例子来说明一些编译器是如何通过指令调度来消除流水线暂停的。有下面连续的两个操作:

$$a = b + c$$

$$d = e - f$$

典型的 DLX 实现上述操作的代码如下。

```
LW      Rb,b
LW      Rc,c
ADD     Ra,Rb,Rc
SW      a,Ra
LW      Re,e
LW      Rf,f
SUB     Rd,Re,Rf
SW      d,Rd
```

进行指令调度之后的代码如下。

```
LW      Rb,b
LW      Rc,c
LW      Re,e;                  交换指令,消除 ADD指令暂停
ADD     Ra,Rb,Rc
LW      Rf,f
SW      a,Ra                   ;Store/Load交换,消除 SUB指令暂停
SUB     Rd,Re,Rf
SW      d,Rd
```

对比指令调度前后的指令序列可以看出,调度前两条 ALU 指令(ADD Ra,Rb,Rc 和 SUB Rd,Re,Rf)分别和两条 Load 指令(LW Rc,c 和 LW Rf,f)之间存在数据相关。要保证流水线能正确执行调度前的指令序列,必须在指令执行过程中插入两个时钟周期的暂停,调度后的指令序列中消除了这样的暂停。在此基础之上,可以通过流水线定向技术消除 ALU 指令和 Store 指令之间的数据相关。

现代的许多编译器都试图通过指令调度来改善流水线的性能。在最简单的算法中,编译器只能调度程序基本块中的指令。所谓程序基本块是指一个线性代码串,除了开始和结尾之外,它没有其他的入口和出口。调度这样的代码串比较容易,因为基本块中的每条指令都是顺序执行的,很容易生成一个指令之间的相关图,并重新排列指令顺序使暂停最小。

4. 数据相关的动态调度

在本章第 3 节和第 4 节中介绍的 DLX 基本流水线中,我们认为流水线负责取指令并发射它,除非已经在流水线中的指令与这个取到的指令之间存在数据相关,而且不能通过定向技术予以避免。定向技术降低了流水线的实际延迟,从而使某些数据相关不会导致冲突。如果有实在不可避免的数据相关,那么检测冲突的硬件将停止相关的指令及其后面的指令进入流水线,直至相关关系清除才会再取出和发射新的指令。为了分离出有相关的指令,减少实际冲突及相应暂停的数目,就需要使用前面介绍由编译器来完成的静态调度。

早期的几种处理器还使用了另外一种方法,即动态调度法。这种方法是由硬件动态调整指令执行顺序以减少暂停的影响。动态调度法不仅能够处理某些在编译阶段无法知道的相关关系(如涉及内存引用时),而且能够简化编译器设计,最重要的是它能够允许在别的流水线机器上编译的指令,在不同的流水线上也能有效地运行。这些优点是以硬件复杂度显

著增加为代价的。

尽管动态调度并不能真正消除数据相关,但它能在出现数据相关时尽量避免出现处理器暂停。相比之下,静态流水线调度方法则是尽量通过分离有相关问题的指令使它们不会导致冲突,从而减少暂停的影响。下面我们就来简单地分析一下动态调度的基本思想。

到目前为止所讨论的流水线技术的一个主要限制,就是它们全都使用按序发射指令机制,即如果指令在流水线中被暂停,那么后继指令也无法前行。因此,若两条相邻的指令存在相关关系,就会导致流水线暂停。对于有多个功能部件的机器,就会造成这些功能部件的闲置。如果指令 j 相关于正在流水线中执行的指令 i,而 i 的运行时间又很长,那么所有 j 后面的指令也必须停下来直至 i 执行完成,j 才能开始执行。例如下面的指令序列:

```
DIVD    F0,F2,F4
ADDD    F10,F0,F8
SUBD    F12,F8,F14
```

ADDD 相关于 DIVD 指令导致流水线暂停,所以 ADDD 不能执行。然而 SUBD 指令与流水线中的所有指令都不相关,它仍然不能执行,这就是由于指令顺序流出而带来的局限性。

在本章第 3 和第 4 节讨论的 DLX 流水线中,结构相关和数据相关都是在指令译码(ID)阶段进行检测的,如果一条指令可以正确执行才会从 ID 发射出去。如果要使上例中的 SUBD 指令开始执行,就必须把发射阶段分为两个阶段:检测结构相关和检测数据相关。在发射指令时仍然能够进行结构相关检测,依旧是使用按序发射指令的方法。但是只要指令的操作数就绪就执行,即指令可以乱序执行,而且指令的结束也是乱序的。

乱序执行最主要的困难是在异常处理上。在采用动态调度方法的处理机中,在某条指令产生异常情况时,有可能出现其后面的指令已经执行完成的情况,这样异常处理是不精确的。这种异常出现之后是难以确定和恢复现场的。精确的异常处理需要使用推断的方法来解决,后面我们将会对推断的方法进行简单介绍。

为了允许乱序执行,将 DLX 基本流水线的译码阶段再分为两个阶段,第一个阶段是发射阶段,完成指令译码,并检测结构相关情况;第二个阶段是读操作数阶段,等待直到不存在数据相关,并读出操作数。在动态调度的流水线中,所有指令在发射阶段都是按序发射的,但读操作数阶段有可能会被暂停或者绕过暂停指令,从而可以乱序执行。

12.4.3 控制相关

在 DLX 流水线中,控制相关对流水线带来的影响比数据相关更大。当执行一条分支指令的时候,它是否适用程序计数器 PC 的值加 4 是不确定的,因为这条分支指令还可能把 PC 改为分支转移成功以后的目标地址。只有当分支转移不会执行的时候,原 PC 的值才会有用。如图 12.20 所示,如果分支指令 i 分支成功转移,那么通常要到 MEM 段的末尾,在已经完成了地址计算和比较之后才能改变 PC。

处理分支指令最简单的方法就是一旦发现分支指令就暂停流水线,即暂停该指令之后

的所有指令,直到分支指令达到MEM段确定了新的PC值为止。当然,我们在发现它是分支指令之前是不愿意暂停流水线的,因此暂停在ID段之后才会发生,图12.31所示的流水线时空图就描述了这种处理方法。从图中可以看出,在流水线中引入了两个暂停周期。控制相关的处理方法和数据相关不同,它要根据新的有效PC值进行取指令操作。这样分支指令上就给流水线带来了3个时钟周期的暂停。

指　令	时　钟									
	1	2	3	4	5	6	7	8	9	10
分支指令	IF	ID	EX	MEM	WB					
分支后继指令		IF	Stall	Stall	IF	ID	EX	MEM	WB	
分支后继指令+1						IF	ID	EX	MEM	WB
分支后继指令+2							IF	ID	EX	MEM
分支后继指令+3								IF	ID	EX
分支后继指令+4									IF	ID
分支后继指令+5										IF

图12.31　简单方法处理分支指令的流水线时空图

如果流水线中处理每条分支指令都要引入3个时钟周期的暂停,那么这必然要严重降低流水线的性能,是对资源一种巨大的浪费。如果流水线理想的CPI是1,目标代码中分支指令占30%,那么分支暂停将使机器大约只能达到流水线理想加速比的一半,所以减少分支的开销就变得十分重要。通常,可以分如下的两个步骤来减少流水线处理分支指令时的暂停周期数。

(1) 在流水线中尽早判断出分支转移是否真正发生。

(2) 尽早计算出分支成功转移时的PC值(如分支的目标地址)。

要对分支指令进行优化处理,就必须解决好上面这两个问题,如果只是计算分支的目标地址,却不知下一条指令到底是由分支目标地址指出的指令,还是由PC+4确定的指令,那么优化就没有实际意义。

在DLX流水线中,分支指令BEQZ和BENZ需要测试分支条件寄存器的值是否为0,可以考虑把测试分支条件寄存器的操作移到ID段完成,这样在ID周期末尾就能够完成分支转移是否成功的检测。实现这种优化的前提还要尽早计算出分支转移成功和分支转移失败时的两个PC值。为此,就需要在ID段增加一个加法器,以避免结构相关。图12.32是对DLX流水线进行上述修改后的数据通路,从中不难看出,改进后的数据通路处理分支指令只需要一个时钟周期的暂停。

在某些流水线机器中,由于检测分支条件和计算分支转移目标地址需要更长的时间,所以处理分支指令时比上面所举的例子要使用更多的时钟周期。一般而言,流水线级数越多,处理分支指令带来的控制相关所需要的时钟周期就越多。

减少流水线分支开销的方法有很多种。前面已经简单讨论了从硬件方面减少流水线暂停周期的方法。下面我们将主要从编译技术的角度,分析几种减少流水线分支开销的简单方法。这几种方法都有一个共同的特点,即这些方法对分支转移是否成功进行的预测都是

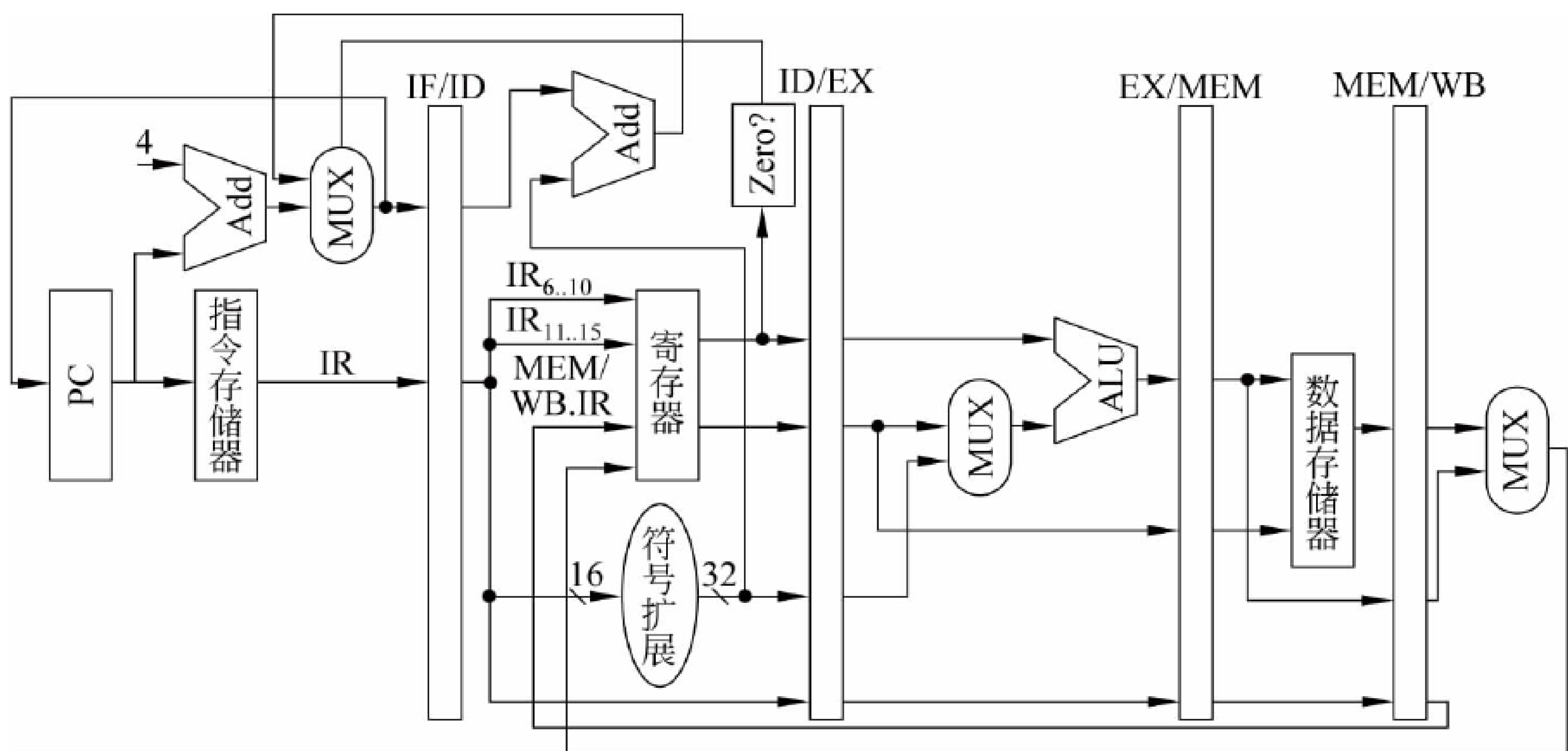


图 12.32　对图 12.20 进行修改后的 DLX 流水线的数据通路

静态的，而且在整个程序的执行过程中保持这种预测结论：要么总是认为分支转移成功，要么总是认为分支转移失败。在下一节我们将讨论更高效的编译器调度方法，例如循环展开能够减少循环分支的频度，此外，在下一节还会涉及一些基于硬件的动态预测调度方法。

1. 冻结或者排空流水线的方法

处理分支最简单的方法就是冻结或者排空流水线，也就是保持或者清除流水线在分支指令之后读入的所有指令，直到知道分支指令的目标地址以及分支转移是否成功为止。这种方法的优点是软件和硬件都很简单。图 12.31 所示的就是这种简单的处理方法，主要应用在早期的流水线中。

2. 预测分支转移失败的方法

DLX 流水线采用预测分支转移失败的方法，这种方法是指当流水线译码到一条分支指令时，就像分支指令就是一条普通的指令那样，流水线继续取后续的指令，并且允许分支指令后续指令在流水线中正常流动。当流水线确定分支转移是否成功以及分支的目标地址之后，如果分支转移成功，流水线就将分支指令之后取出的所有指令用空操作来代替，同时从分支指令目标地址处取来有效指令继续执行；如果分支转移失败，那么流水线仍然可以正常流动，分支指令就看作一条普通的指令一样，也就不需要废弃分支指令之后取出的所有指令。图 12.33 所示的就是采用这种方法处理分支指令的流水线时空图。

3. 预测分支转移成功的方法

和预测分支转移失败相反的另外一种方法就是预测分支转移成功，采用这种方法时，一旦完成分支指令的译码并且计算出了分支的目标地址，就假设分支转移成功，并且开始在分支目标地址处取指令执行。

4. 分支延迟的方法

分支延迟技术是一种软件方法，它由编译程序重排指令序列来实现。其基本思想是从逻辑上“延长”分支指令的执行时间，即发生“分支转移成功”时并不排空指令流水线，而是让紧跟在分支指令 I 之后已进入流水线的少数几条指令继续完成，如果这些指令是与分支指令 I 结果无关的有用指令，那么延迟损失时间就可以被有效地利用。

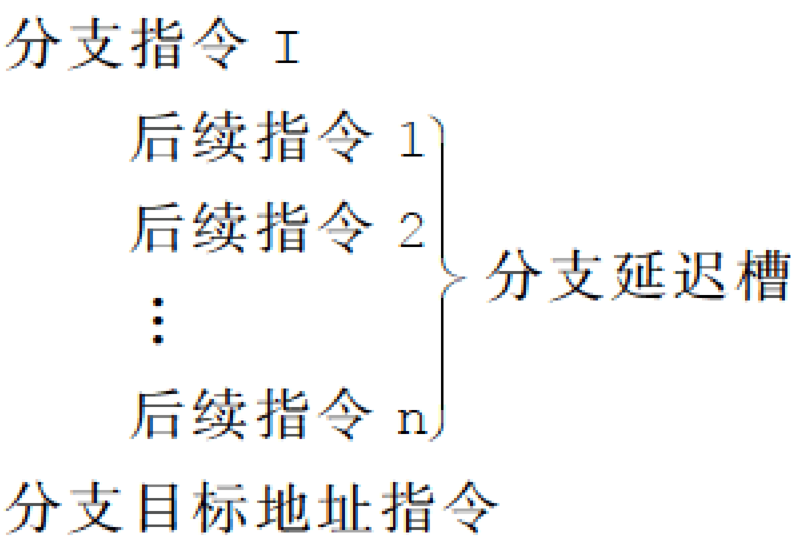
指　令	时　钟								
	1	2	3	4	5	6	7	8	9
分支指令 i	IF	ID	EX	MEM	WB				
指令 $i+1$		IF	ID	EX	MEM	WB			
指令 $i+2$			IF	ID	EX	MEM	WB		
指令 $i+3$				IF	ID	EX	MEM	WB	
指令 $i+4$					IF	ID	EX	MEM	WB

(a) 分支转移失败

指　令	时　钟								
	1	2	3	4	5	6	7	8	9
分支指令 i	IF	ID	EX	MEM	WB				
指令 $i+1$		IF	Idle	Idle	Idle	Idle			
分支目标			IF	ID	EX	MEM	WB		
分支目标+1				IF	ID	EX	MEM	WB	
分支目标+2					IF	ID	EX	MEM	WB

(b) 分支转移成功

图 12.33　采用预测分支转移失败方法时流水线时空图



后续指令放在分支延迟槽中,不管分支转移是否成功,这些指令都要被流水执行。并不是所有的指令都能够放到延迟槽中,选择放到分支延迟槽中的指令必须按照一定的原则经过编译器的调度。图 12.34 画出了对分支延迟的 3 种调度方法,表 12.6 则给出了这 3 种调度方法所受的限制,以及它们各自的应用场合,显然采用这种分支延迟方法可以减少流水线分支开销。

5. 控制相关的动态调度

解决控制相关问题不仅对于每个周期发射一条指令的处理机能够起到有效的作用,而且对于后面要讲到的单周期发射多条指令的处理机也是很重要的。在单周期发射 n 条指令的情况下,分支操作将会以超过 n 倍的情形出现,这更要求对分支情况提前做出预测。

在本节前面部分给出了处理分支的各种静态调度方法,这些方法不受分支动态行为的影响。还讨论了分支延迟机制,它允许在编译阶段通过软件手段来调度和优化分支。除了软件方法之外,还可以通过硬件动态地进行分支处理,对程序运行时分支的行为进行预测,

提前对分支操作做出反应，提高分支的处理速度。下面对分支预测缓冲技术、分支目标缓冲技术和推断执行技术进行简单的介绍，更详细的资料可以参考有关文献。

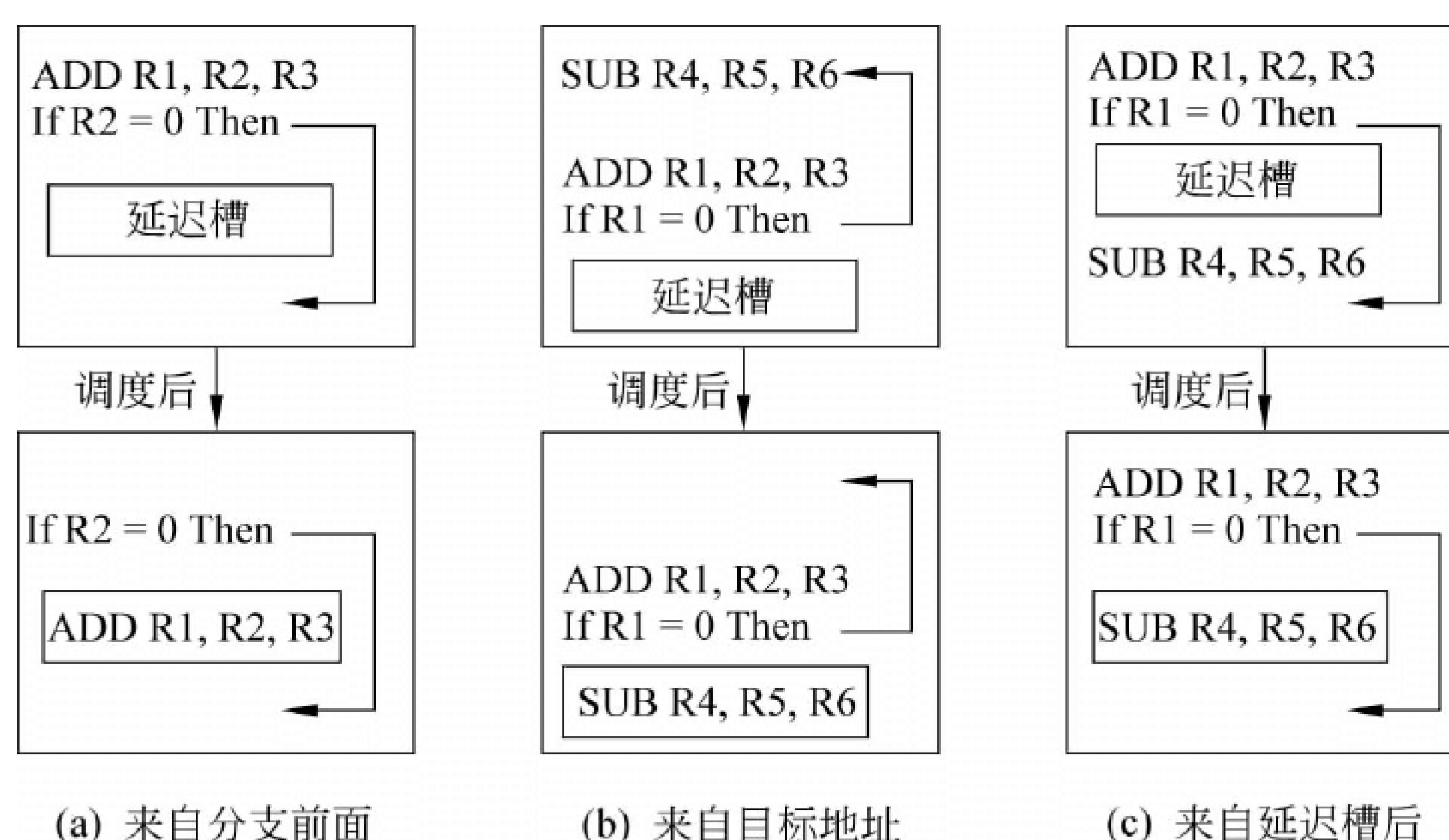


图 12.34 分支延迟的 3 种调度方法

表 12.6 分支延迟的几种调度方法以及它们的限制

调度策略	所受限制	对流水线性能改善的影响
(a)来自分支前面	被调度指令必须与分支指令不相关，即分支必须不依赖于被调度的指令	总是可以提高流水线的性能
(b)来自目标地址	如果分支转移失败，必须保证被调度的指令对程序的执行没有影响，可能还需要复制被调度的指令	分支转移成功时，可以提高流水线的性能。但由于需要复制指令，可能加大程序空间
(c)来自延迟槽后	如果分支转移成功，必须保证被调度的指令对程序的执行没有影响	分支转移失败时，可以提高流水线的性能

最简单的动态分支预测方法（Dynamic Branch Prediction）是使用分支预测缓冲区（Branch Prediction Buffer，BPB）。缓冲区的每一项内容被用来预测分支转移是否成功，并且根据实际的分支情况对内容进行修改。这种方法是基于如下的考虑：如果本次分支转移成功了，那么预测下一次分支转移也成功。

改进的动态分支预测方法是分支目标缓冲区（Branch Target Buffer，BTB）技术。具体做法就是将分支转移成功的分支指令的地址和它的分支目标地址都放到一个缓冲区中保存起来，缓冲区以分支指令的地址作为标志；在取指令阶段，所有的指令地址都与保存的标志作比较，如果相同，就认为本条指令是分支指令，而且认为它分支转移成功，同时它的分支目标（下一条指令）地址就是保存在缓冲区中的分支目标地址。

推断执行是指在处理器还没有判断指令是否能够执行之前就提前执行的一种技术，采用这种技术可以有效地克服控制相关。这种推断执行的过程带有明显的投机性质，如果推断准确，它可以消除所有附加延迟。因此在大多数推断准确的情况下，推断执行技术可以有效地加快分支处理速度。基于硬件方案的推断执行技术可以说是根据动态的数据相关性来选择指令的执行时刻，它综合了以下想法：动态的分支预测决定执行哪条指令；在控制相关消除之前推断执行指令；对程序基本块采用指令动态调度。

实现推断的最重要的思想就是允许指令乱序执行但顺序确认,只有确认以后的结果才是最终的结果,从而避免如更新状态或者发生异常等不可恢复的行为。但是推断技术也存在着一个主要的缺点,即支持推断技术的硬件太复杂,需要大量的硬件资源。

12.5 指令级并行技术

12.5.1 基本概念

从本章前面几节可以知道,当指令不相关时,它们在流水线中是重叠执行的。这种指令序列中存在的潜在并行性称为指令级并行。下面主要讨论的就是如何通过各种可能的技术,获得更多的指令级并行性,即需要开发指令级并行度(Instruction Level Parallelism,ILP),它定义为在一个时钟周期内流水线上流出的指令数。

衡量指令级并行性的一个指标是(Clock Cycles Per Instruction,CPI),它定义为流水线中执行一条指令所需的时钟周期数。在前面已经介绍了,理想的流水线性能是每一个时钟周期都能启动一条指令的执行。假设流水线有 m 个流水段,一个程序执行时共执行了 n 条指令,则平均每条指令所占用的时钟周期数 $CPI=(m+n-1)/n$,当 $n>>m$ 时,$CPI\approx1$。为了达到 CPI=1 这样的理想情况,必须采用相应的技术减少数据相关和控制相关,在上一节中已经介绍了处理相关问题的基本方法。

指令流水处理机比传统的串行处理机有较高的吞吐率,其原因在于多条指令可在流水线的不同功能段中同时进行操作,即实现了指令级的并行性。但要进一步提高流水线的吞吐率,获得更高的性能,就必须使 CPI<1,那么有没有办法使 CPI 小于 1 呢?原来我们不能使 CPI<1 的最直接原因就是每个时钟周期只能流出一条指令,如果流水线在一个时钟周期内能够有多条指令流出的话,目的就可以实现了,多指令发射处理器就有效地解决了这个问题。多指令发射处理器主要有 4 种,即超标量(Superscalar)、超流水线(Superpipelining)、超标量超流水线(Superscalar Superpipelining)和超长指令字(Very Long Instruction Word,VLIW)。

12.5.2 多指令发射技术

超标量、超流水线和超标量超流水线 3 种处理机在一个时钟周期内可以执行完成多条指令,即它们的 ILP>1。如果用一台 k 段流水线的普通标量流水处理机做比较基准,为了便于比较,把基准标量处理机的机器流水线周期和指令发射等待时间都假定为 1 个时钟周期,同时发射的指令条数为 1 条,即假设它的指令级并行度 ILP 为 1。超标量处理机的并行度为 m,超流水线处理机的并行度为 n,超标量超流水线处理机的并行度为(m,n),则 4 种不同类型处理机的时空图和 ILP 如图 12.35 所示,其主要性能比较如表 12.7 所示。

在目前的微处理机中,大多数属于超标量处理机。例如,Intel 公司的 i860、i960、Pentium,Motorola 公司的 MC88110,IBM 公司的 Power 6000,SUN 公司的 Super SPARC 等都是超标量处理机。SGI 公司的 MIPSR 4000、R5000、R10000 等都是超流水线处理机。DEC 公司的 Alpha 是超标量超流水线处理机。

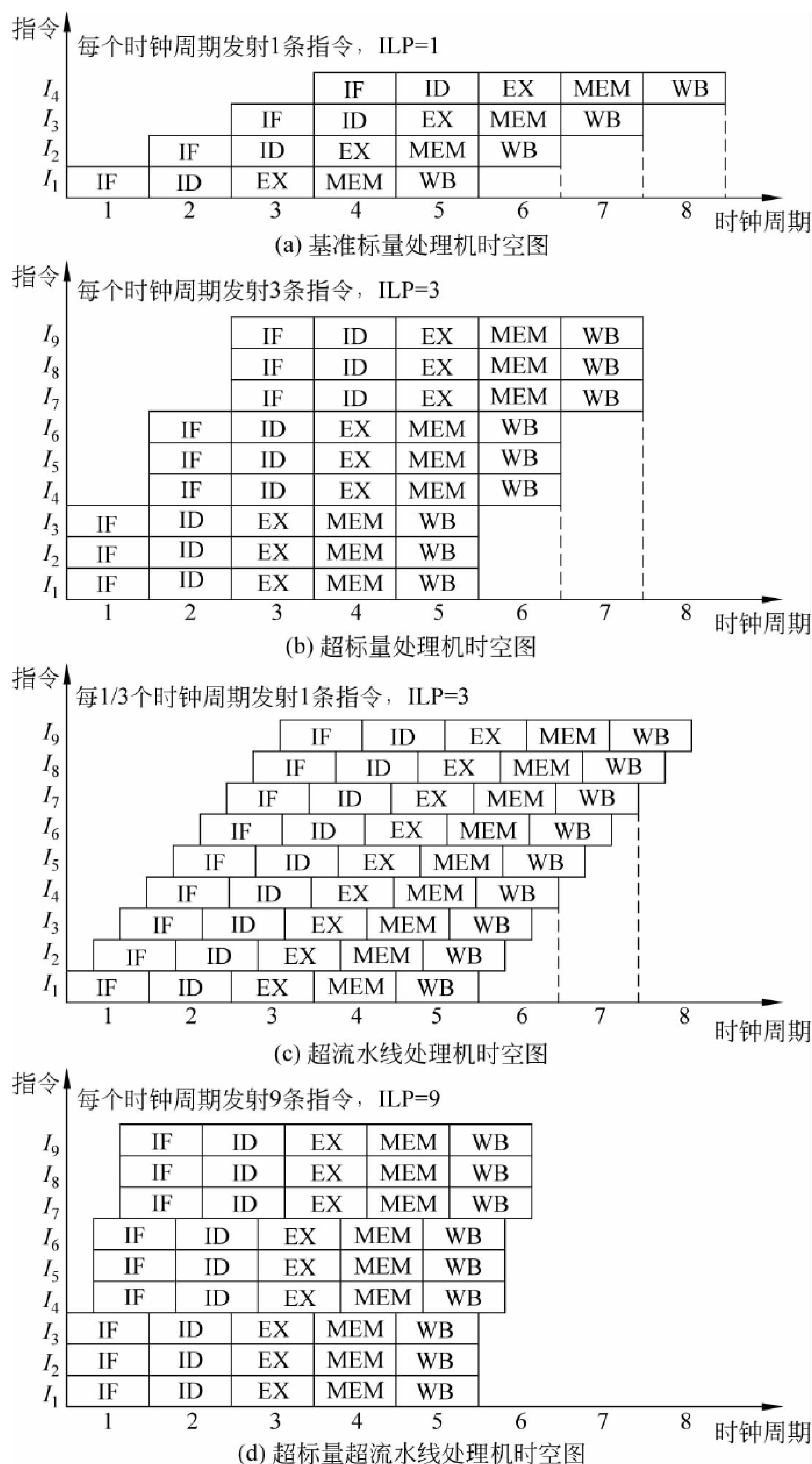


图 12.35　几种不同处理机的时空图和 ILP

表 12.7　4 种不同类型处理机的性能比较

	普通标量处理机	超标量处理机	超流水线处理机	超标量超流水线处理机
机器流水线周期	1 个	1 个	$1/n$ 个	$1/n$ 个
同时发射指令数	1 条	m 条	1 条	m 条
指令级并行度	1	m	n	$m \times n$

1. 超标量处理机

通常把一个时钟周期内能够同时发射多条指令的处理机称为超标量处理机。这种处理机的基本要求是必须要有两套或者两套以上的完整的指令执行部件。图12.35(b)是典型的超标量处理机的指令流水线时空图。为了能够在一个时钟周期内同时发射多条指令,超标量处理机必须有两条或者两条以上能够同时工作的指令流水线。高性能超标量处理机一般还有一个先行指令窗口,它能够从指令Cache中预取多条指令,而且能够对这些指令进行数据相关性分析和功能部件冲突检测。这些同时也能够说明超标量流水线处理器的控制逻辑是相当复杂的。

超标量处理机是开发空间并行性,在每个时钟周期可以平均执行完成多条指令。如果一台超标量处理机每个时钟周期同时发射 m 条指令,则它的指令级并行度ILP的期望值就为 m。但由于资源冲突、数据相关、控制相关等原因,实际的ILP不可能达到 m,通常是 $1<ILP<m$。

2. 超流水线处理机

一般把在一个时钟周期内能够分时发射多条指令的处理机称为超流水线处理机。另外,也把指令流水线的段数大于等于8的流水线处理机称为超流水线处理机。超流水线处理机和超标量处理机的工作方式不同,超标量处理机是通过重复设置多个部件,并且让这些部件能够同时工作来提高指令的执行速度,实际上是以增加硬件资源为代价换取处理机性能的;而超流水线处理机则只是通过增加少量硬件,通过各部分硬件的充分重叠工作来提高处理机的性能。

从图12.35(c)可以看出,超标量处理机采用的是空间并行性,而超流水线处理机是开发时间并行性,通过各部分硬件的充分重叠来提高机器性能。一台并行度ILP为 n 的超流水线处理机,它在一个时钟周期内能够发射 n 条指令。但是 n 条指令不是同时发射,而是每隔 $1/n$ 个时钟周期发射一条指令。

3. 超标量超流水线处理机

从指令级并行性来看,超标量处理机主要开发空间并行性,依靠重复设置的操作部件同时执行多个操作来提高程序的执行速度。而超流水线处理机则主要开发时间并行性,在同一个操作部件上重叠多个操作,通过使用较快时钟周期的深度流水线来加快程序的执行速度。

为了进一步提高指令级并行度,可以把超标量技术与超流水线技术结合在一起,这就是超标量超流水线处理机。图12.35(d)表示它的指令执行时空图。它在一个时钟周期内要发射指令 m 次,每次发射指令 n 条,故每个时钟周期中总共发射指令 $m\times n$ 条。

超标量超流水线处理机既开发空间并行性,又开发时间并行性。其并行度期望值为 $m\times n$。

4. 超长指令字处理机

超长指令字处理机主要是基于以下思路:由编译程序在编译时找出指令间潜在的并行性,进行适当调度安排,把多个能并行执行的操作组合在一起,成为一条具有多个操作段的超长指令。由这条超长指令去控制VLIW处理机中多个互相独立工作的功能部件,每个操作段控制一个功能部件,相当于同时执行多条指令。

VLIW处理机是一种单指令多操作码多数据的系统结构。VLIW的字长与机器中的执

行部件数有关。一般来说，对于每一个执行部件需要有一个长度为16～32位的操作段，因此VLIW处理的指令字长度约在100～1000位之间。典型的机器有Cydrome公司Cydra 5(1989年)，飞利浦公司的TM-1(1996年)。VLIW处理机用一条长指令实现多个操作的并行执行，以减少对存储器的访问。并行操作主要是在流水的执行阶段进行的，如图12.36所示，在执行阶段可并行执行3个操作，相当于指令级并行度为3。

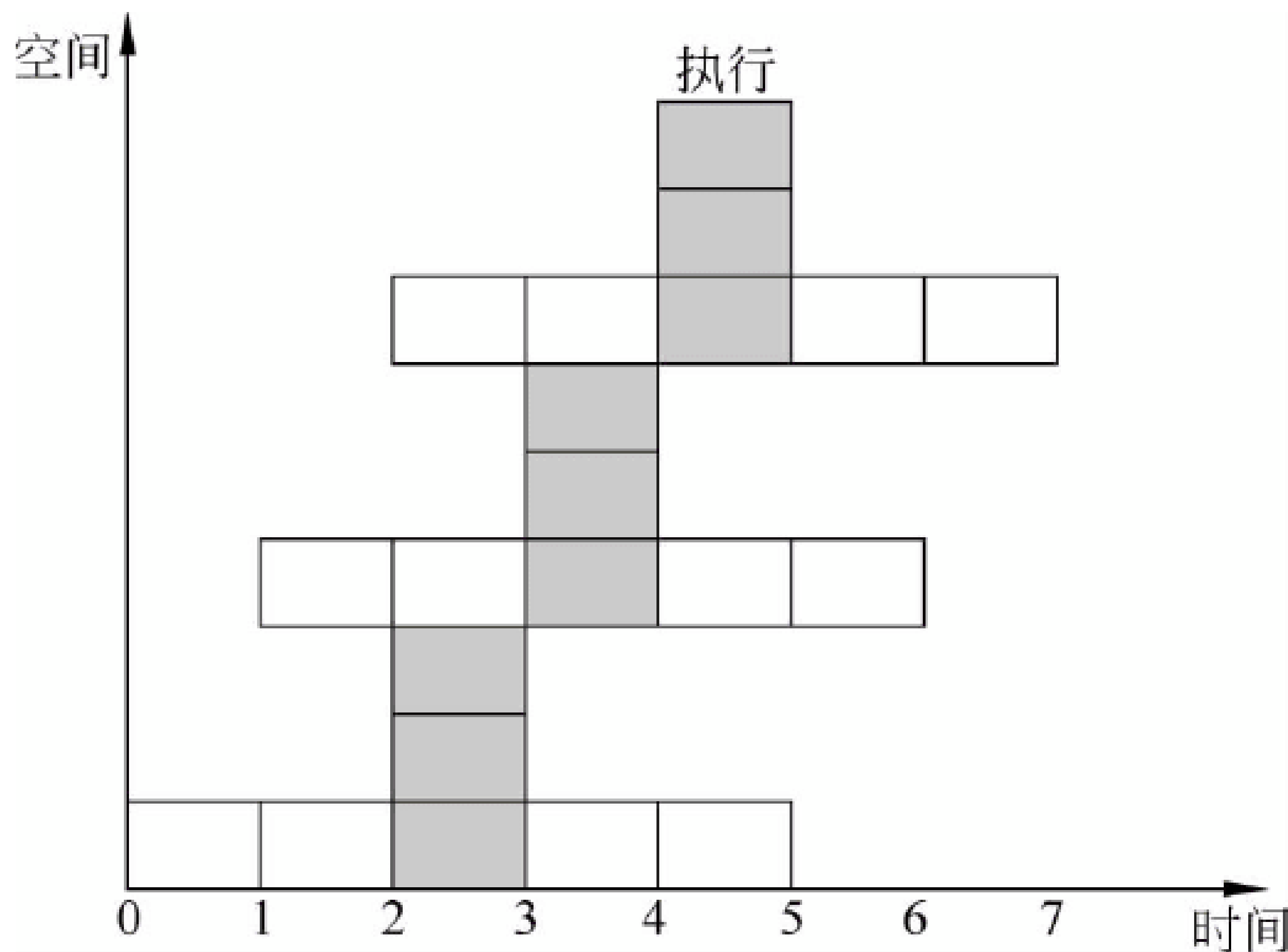


图12.36　VLIW处理机时空图

VLIW处理机具有以下主要特点：超长指令字的生成是由编译器来完成的，由它将串行的操作序列合并为可并行执行的指令序列，以最大限度实现操作并行性；采用单一的控制流，只设置一个控制器，每个时钟周期启动一条长指令；超长指令字被分成多个控制字段，每个字段直接独立地控制每个功能部件；含有大量的数据通路和功能部件，由于编译器在编译时间已解决可能出现的数据相关和资源冲突，所以控制硬件比较简单。

本章内容小结和学习方法建议

流水线技术是一种经济、有效的时间并行技术。在现代计算机设计中得到了最广泛的应用。常用的流水线表示方法有连接图和时空图，衡量流水线的主要性能指标有加速比、吞吐率和效率。

流水线设计的一个关键问题是要保证流水线能畅通流动，阻碍流水线畅通的主要因素有结构相关、数据相关和控制相关。解决上述3类相关最简单的方法是使流水线暂时停顿，直至相关条件消失为止，但是这将使流水线的性能有较大损失。减少或消除结构相关影响的方法主要是采用资源重复方法；消除数据相关的有效方法是设置专用通路以尽快将运算结果直接送到使用它的地方；消除和减少控制相关的方法，主要是加速生成转移目标地址和采用延迟转移技术。

尽管指令流水线技术得到普遍应用，也很重要，但从教学的角度看，它在课程中仍然属于提高性的内容，对待基础性知识和提高性的内容有一个合理分配精力的问题。我们建议，学生首先要比较准确地掌握多指令周期CPU的组成和指令执行过程，理解指令流水线能够为系统带来高性能和更好性价比的理由，了解它带来的问题和解决问题的思路，对那些精力有点不够的学生来说，就不必深究解决3类相关问题的具体方案与技术。

指令级并行技术是指细粒度并行性,它不仅包括时间并行技术,还包括空间并行技术。衡量指令级并行性的一个指标是CPI,为了得到理想的CPI就要减少流水线中的各种相关,如果要想使CPI能够低于1,可以采用多指令发射技术,即在一个时钟周期内发射多条指令,对此只要从概念上理解即可,不必深究。

习题与思考题

1. 指令执行过程采用顺序方式、一次重叠方式和流水线方式,它们的主要差别是什么?各有什么优点和缺点?

2. 在指令流水线中,每一条指令执行过程的时间减少了吗?如果没变减少,那么为什么还要采用流水线技术呢?一般来说流水线有哪些特点?总结流水线的各种分类方法的分类原则。

3. 描述CPU时常用CPI(每条指令平均时钟周期数)、MIPS(每秒百万条指令数)、MFLOPS(每秒百万次浮点操作数)这样的量来表征。请写出它们的定义式,并求出MIPS、CPI与CPU时钟频率f的关系。

4. 假设一条指令的执行过程分为"取指令"、"分析"和"执行"3段,每段的时间分别是Δt、$2\Delta t$和$3\Delta t$。在下列各种情况下,分别写出连续执行n条指令所需要的时间表达式。①顺序执行;②仅"取指令"和"执行"重叠;③"取指令"、"分析"和"执行"重叠。

5. 有一条由4个功能段组成的流水线,每个功能段都使用1个时钟周期,周期长度为Δt。每输入5条指令后停顿2个时钟周期,求此流水线的实际加速比、吞吐率和效率。

6. 一条线性静态多功能流水线由6个功能段组成,加法操作使用其中1、2、3、6功能段,乘法操作使用其中1、4、5、6功能段,每个功能段的延迟时间都相等。流水线的输入端和输出端之间有直接数据通路,而且设置有足够的缓冲寄存器。用这条流水线计算$F=\sum_{i=1}^{6}(A_i \times B_i)$,画出流水线时空图,并计算流水线的实际吞吐率、加速比和效率。

7. 流水线有m段,各段的延迟时间分别是$t_i(i=1,2,\cdots,m)$,现有n个任务需要完成,并且每个任务都需要流水线各段实现,请计算:①流水线完成这n个任务所需要的时间;②和非流水线实现相比,这n个任务流水实现的加速比是多少?加速比的峰值又是多少?

8. 什么是流水线的相关问题?通常都有哪几类相关问题?这些相关问题都是什么原因造成的?各种相关问题都有哪些解决方法?

9. 一条有3个功能段的流水线,每个功能段的延迟时间都相等,为Δt。其中功能段2的输出要返回它自己的输入端循环一次,然后再流到功能段3。请问:①如果每隔一个Δt向流水线的输入端连续输入任务,这条流水线会发生什么情况?②求这条流水线能够正常工作的最大吞吐率、加速比和效率。③有什么方法能够提高这条流水线的吞吐率?

10. 假设分支目标缓冲的命中率是90%,程序中无条件转移指令的比例是5%,没有无条件转移指令的程序CPI为1。假设分支目标缓冲中包含分支目标指令,允许无条件转移指令进入分支目标缓冲,则程序的CPI值是多少?

11. 什么是多指令发射技术?多指令发射有几种方式?每种方式的特点是什么?

第13章 并行计算机体系结构

随着计算机速度的提高，人们对计算机性能的要求也越来越高。因此，为了使计算机能够处理越来越复杂的问题，计算机体系结构设计者把注意力转向了并行计算机。本章从计算机体系结构的发展和分类入手，讨论了计算机体系结构不同层次的并行性，重点对并行计算机系统设计中的主要问题进行了分析，并且在此基础上介绍了 SIMD 和 MIMD 并行计算机等相关内容。在并行计算机体系结构领域内，人们已经进行了大量的研究工作，本章只能讨论其中的一小部分。

13.1 计算机体系结构概述

13.1.1 计算机体系结构的发展

在本书的第 1 章中已经概要介绍了计算体系结构的发展情况，从世界上出现第一台计算机到现在的 50 多年里，计算机体系结构已经取得了重大的进展，但大多数的计算机体系结构仍然没有摆脱冯·诺依曼机器结构的范畴。

随着技术的进步，当前计算机体系结构主要是沿着两个方向发展。第一个发展方向是改变冯·诺依曼机器的串行执行模式，第二个发展方向是改变冯·诺依曼机器的控制驱动方式。就当前的发展状况而言，第一种发展方向，即控制驱动方式下并行处理体系结构的计算机，已经取得了重大进展和一系列的成果。不论是硬件技术还是相应的软件技术都已经相当成熟，并且有很多产品走向市场，获得了广泛的应用，它们代表了当前计算机体系结构发展的主流。而第二种发展方向，除了数据流计算机已经有了一些成型的计算机之外，大多数还属于探索、研究阶段，还需要进行大量的工作。本章的重点就在于介绍计算机体系结构的第一种发展方向的相关成果。

可以在计算机体系结构的不同层次引入并行机制。在最底层，可以通过流水线和使用多个功能单元的超标量设计将并行加入到 CPU 芯片，可以通过使用隐式并行的超长指令字来加入并行，也可以在 CPU 中加入一些特殊的特性来同时控制多线程，甚至将多个 CPU 放到同一个芯片里组成多核处理器。接下来一个层次，可以将具有额外处理能力的附加 CPU 加入到系统中。通常，这些插件 CPU 都具有特殊的功能，例如网络分组处理、多媒体处理或者加密解密等。将并行机制引入上述两个层次一般能够把计算机的性能提高 5～10 倍。

然而,如果想要将性能提高百倍、千倍甚至数万倍,就必须使用多个CPU,让它们一起高效地工作,这种思想造就了大规模多处理器和多计算机系统。毫无疑问,将成千上万的处理器连接成一个大的系统必然会带来需要解决的许多新问题。

当两个CPU或者处理元件紧密连在一起的时候,它们之间具有高带宽和低延时,而且是亲密计算,此时称它们为紧密耦合。相反,当它们间隔较远,具有低带宽高延时,而且是远程计算,此时称它们为松散耦合。在大多数情况下,松散耦合的系统适于解决粗粒度的问题,而紧密耦合的系统适于解决细粒度的问题。图13.1给出了不同层次并行技术的图示。

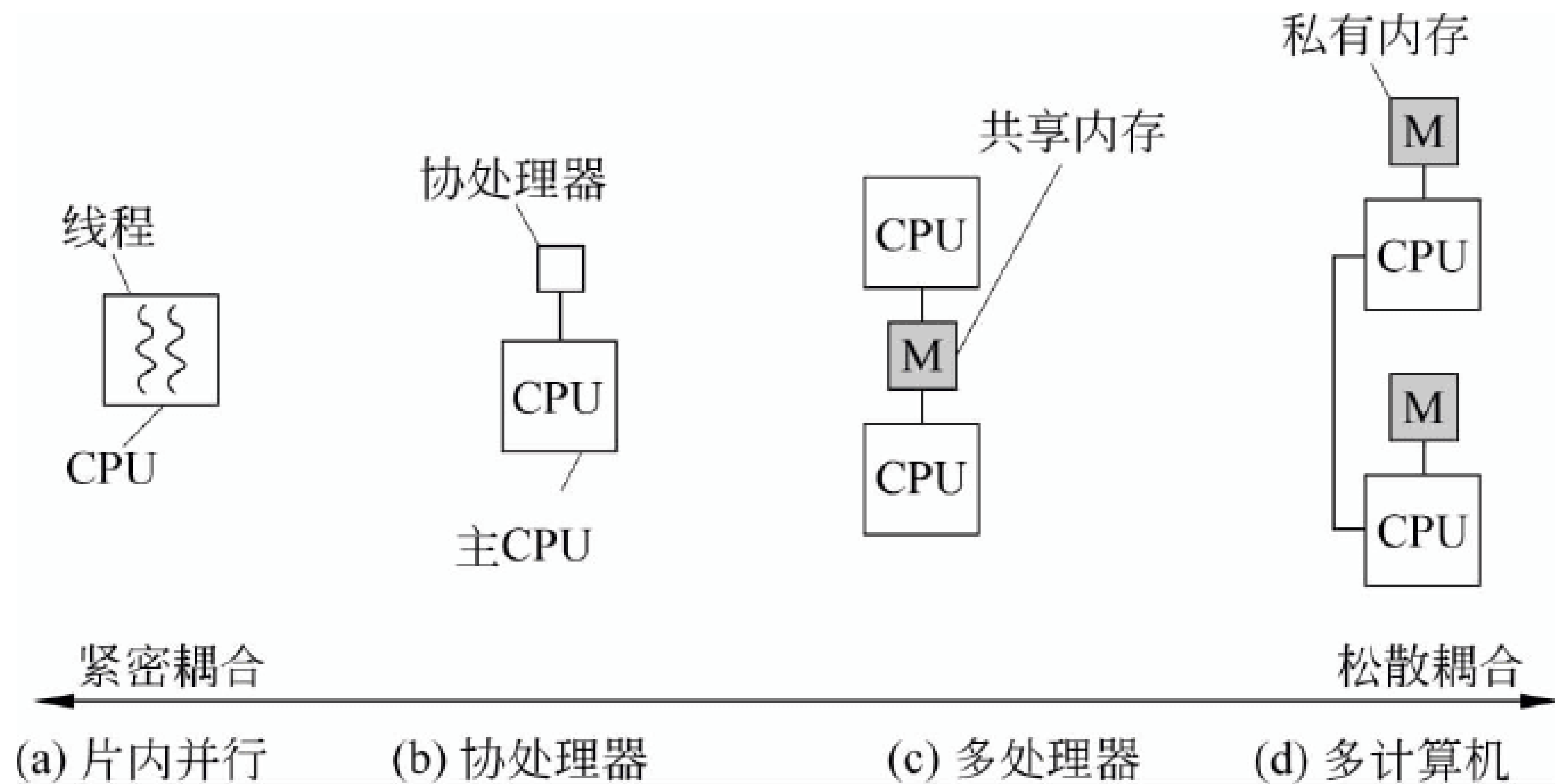


图13.1 计算机体系结构不同层次的并行性

并行计算机系统的研究之所以在近年来取得了重大的进展,不仅从几台或几十台处理器组成的简单系统发展到成百上千台处理机组成的大规模并行系统,而且从科研单位、政府部门专用的并行计算机发展到走向商品销售领域的商用并行计算机系统,其中主要的原因有如下的几个方面。

首先,随着计算机速度的提高,人们对计算机性能的要求也越来越高。天文学家希望使用计算机来模拟整个宇宙演化的过程,药物学家希望使用计算机设计用于治疗特殊疾病的药品,飞机设计师总是希望能设计出更加节约燃料的飞机。总而言之,无论现在计算机的能力多么强大,对于许多用户来说,尤其是把计算机用于科学计算、工程和工业设计的用户来说,仍然远远不够。

其次,大量商品化的处理器的出现为设计并行计算机系统提供了可能。当代的并行计算机系统大都以PC机或者工作站这样的单个处理机作为系统的基本构件,面向商业销售的具有较高性能价格比的RISC处理器产品的大量出现,为并行计算机设计提供了较大的选择余地,也为系统性能的全面提高提供了坚实的基础。

最后,并行计算机系统获得快速发展和处理机间通信技术的发展密不可分。并行计算机系统中大量的多处理机要能够高效地协同工作,处理机之间的通信是至关重要的。它们或者以共享内存变量或者以消息传递机制来进行通信。各种互联网络技术现在已经相当成熟,从理论上和技术上为并行计算机系统的发展准备了前提条件。

13.1.2 计算机体系结构的分类

过去曾普遍将计算机系统分为巨、大、中、小、微型机5类,这是按照规模、性能、速度以及价格的一种大致划分。这种划分计算机系统的方法只能对同时期的计算机大致分类,而

各种计算机系统的规模、价格，尤其是性能和速度的指标随着时间的变化而变化。而且这种分类方法也不能反映计算机的体系结构特征。

1966 年，Michael. J. Flynn 提出按指令流和数据流的多倍性对计算机体系结构进行分类。由于当前的计算机体系结构主流发展方向是控制驱动方式下的并行处理，因此这一分类法获得了普遍的赞同，虽然这种分类法也是非常粗略的。表 13.1 是 Flynn 分类法。

表 13.1　计算机体系结构的 Flynn 分类法

指令流	数据流	名称	举　　例
1 个	1 个	SISD	传统的冯・诺依曼计算机
1 个	多个	SIMD	超级向量处理机，阵列处理机
多个	1 个	MISD	目前还没有
多个	多个	MIMD	多处理机，多计算机

Flynn 分类法是基于指令流（Instruction Stream）和数据流（Data Stream）这两个概念的。指令流是指机器执行的指令序列；数据流就是由指令流调用的数据序列，包括输入数据和中间结果。而多倍性是指在系统最受限制的部件上，同时处于同一执行阶段的指令或数据的最大数目。从某种程度上说，指令流和数据流是互相独立的，因此一共存在 4 种组合。图 13.2 分别表示了它们的基本结构（不包括 I/O 设备）。

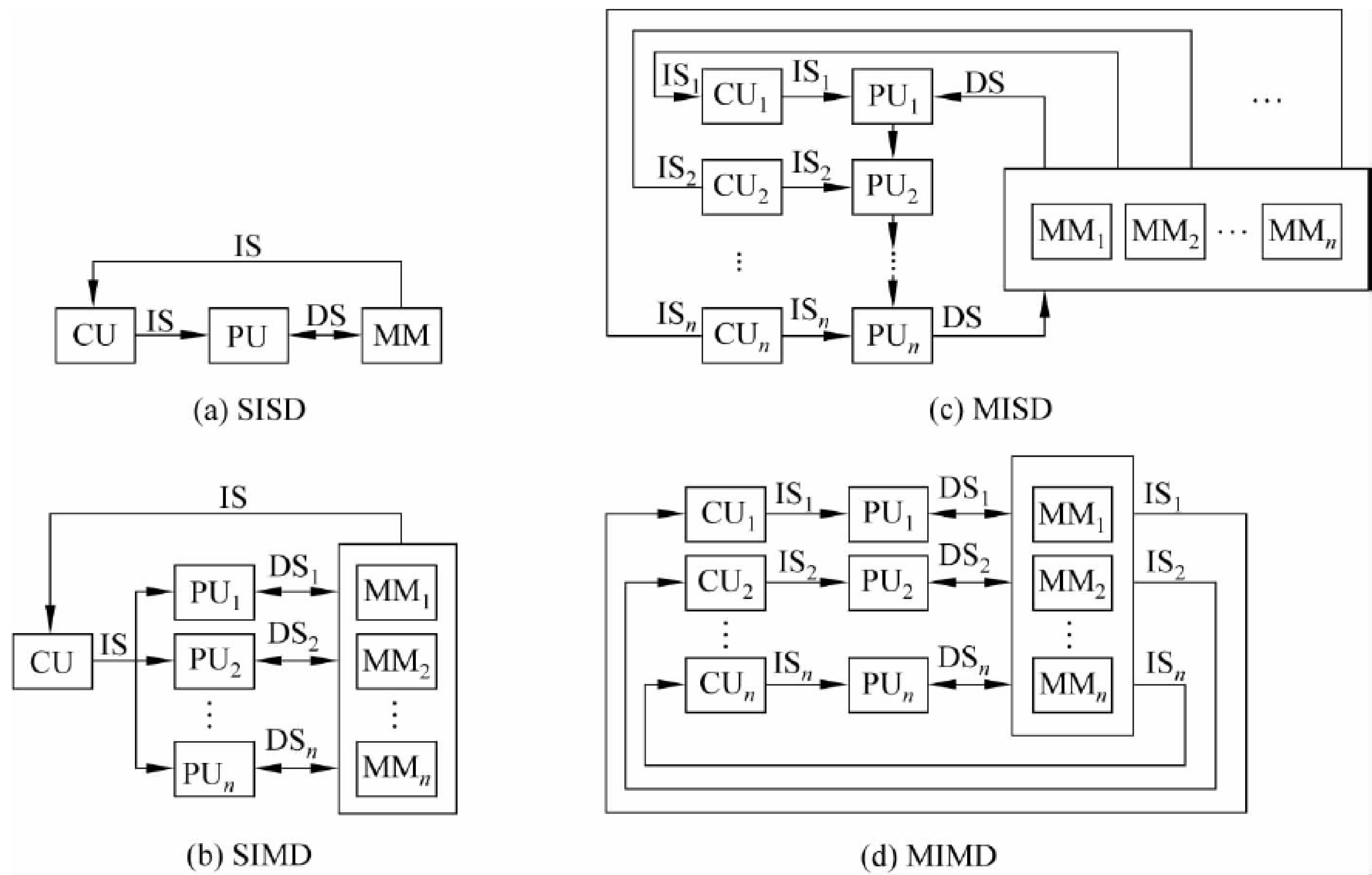


图 13.2　Flynn 分类法各类机器结构

CU：控制部件；PU：处理部件；MM：存储器模块；IS：指令流；DS：数据流

1. SISD 体系结构

这种体系结构代表了传统的冯・诺依曼机器，即大多数的单机（处理器）系统。处理器串行执行指令或者处理器内采用指令流水线，以时间重叠技术实现了一定程度上的指令并行执行；甚至于处理器是超标量处理器，内有几条指令流水线实现了更大程度上的指令并行

执行。但它们都是以单一的指令流从存储器取指令,以单一的数据流从存储器取操作数和将结果写回存储器。

2. SIMD 体系结构

这种体系结构有单一的控制部件,但是有多个处理部件。计算机以一个控制单元从存储器取单一的指令流,一条指令同时作用到各个处理单元,控制各个处理单元对来自不同数据流的数据组进行操作。这种体系结构的典型代表是阵列处理机,一些学者认为将向量处理机也划入此类。值得一提的是当前的很多种处理器都具有多媒体指令功能,如 Intel 的 Pentium Ⅱ/Ⅲ的 MMX 指令等。这类指令能够对打包数据中的多个数据元素同时进行操作。这是 SIMD 的一种变异,通常被简称为 SIMD 类指令。

3. MISD 体系结构

这种体系结构中,有几个处理部件,各配有相应的控制部件。各个处理部件接收不同的指令,多条指令同时在一份数据上进行操作。这种计算机体系结构是一种比较奇怪的组合,这已经被证明是不可能至少是不实际的,目前为止还不存在这种类型的计算机。

4. MIMD 体系结构

这种体系结构中,同时有多个处理部件,并且每个处理部件都配有相应的控制部件。各个处理部件可以接收不同的指令并对不同的数据流进行操作。大多数现代的并行计算机都属于这一类。多处理机系统和多计算机系统都是 MIMD 型的计算机。

总之,Flynn 分类法能够反映大多数计算机系统的并行性、工作方式和结构特点,但它分类的对象主要是控制驱动方式下的串行处理和并行处理计算机。对于非控制驱动方式的计算机,如数据流计算机,就不能采用 Flynn 分类法。正是因为这些原因,其他的学者也提出了一些其他的分类法,如美籍华人冯泽云教授在 1972 年提出了按最大并行度来定量描述各种计算机系统的冯氏分类法;又如 Wolfgan Handler 在冯氏分类法的基础上,于 1977 年根据并行度和流水线提出了另外一种分类法。此外还有其他很多分类方法,这里就不详细介绍了,读者可以参阅有关资料。

13.1.3 并行计算机体系结构分类

依据计算机体系结构 Flynn 分类法,一般来说可以将并行计算机系统分为 4 类,即 SIMD 阵列处理机、SIMD 向量处理机、MIMD 多处理机和 MIMD 多计算机。图 13.3 是按照 Flynn 分类法归纳的并行计算机体系结构图谱。

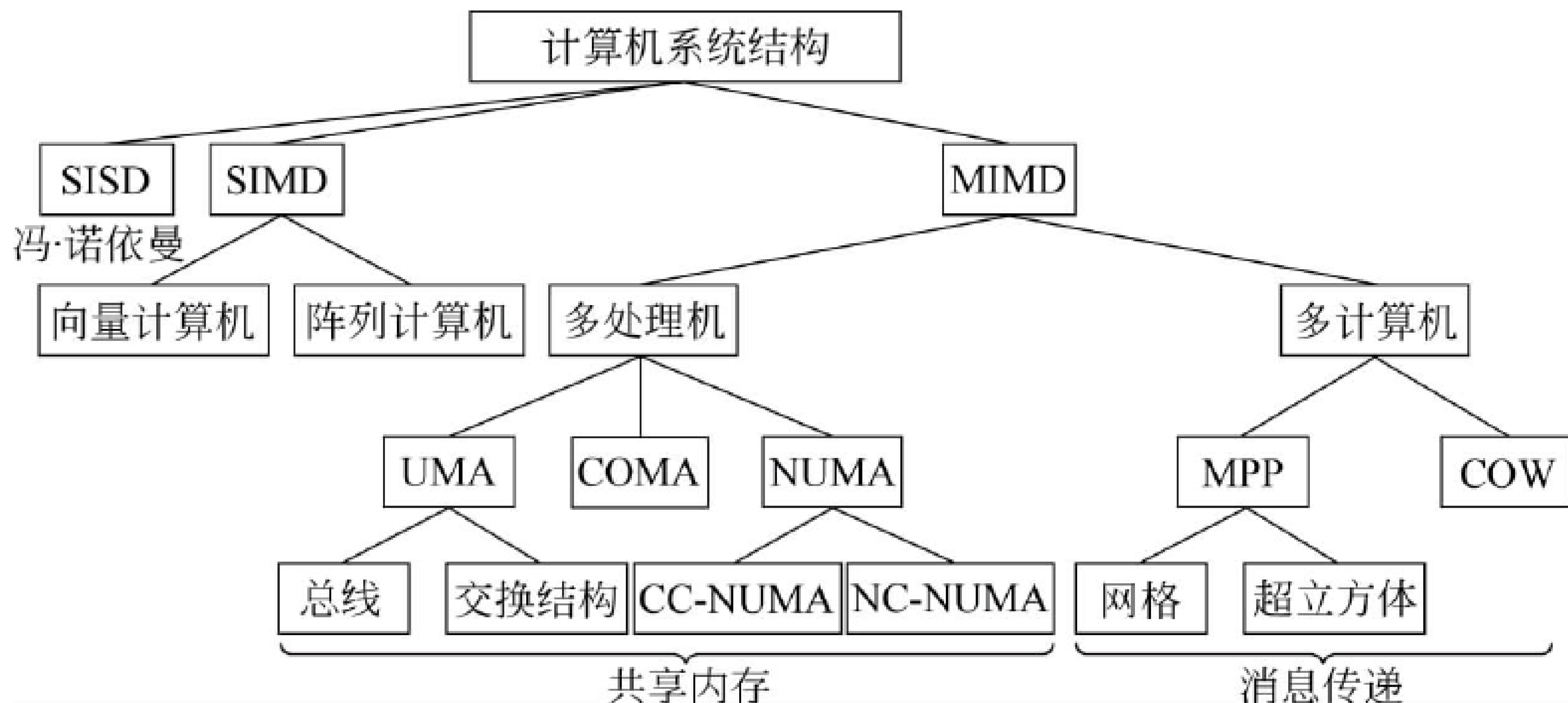


图 13.3 并行计算机的分类

SIMD 体系结构可以分成两个子类。第一类是用于数值计算的超级计算机和其他一些向量处理机，它们可以在一个向量的每个元素上执行相同的操作。一般来说向量处理机是以流水线式 ALU 为核心，实现向量各个元素的并行计算采用的是时间重叠技术。第二类是处理并行类型的阵列处理机，这类计算机采用资源重复方法引入空间因素，即在系统中设置多个相同的处理单元来开发并行性。此外，它是利用并行性中的同时性，所有处理单元必须同时进行相同操作。

在图 13.3 的分类中，我们把 MIMD 体系结构分成了多处理机系统和多计算机系统，前者是基于共享内存的计算机系统，而后者是易于消息传递的计算机系统。

因为在大多数 MIMD 多处理机系统中内存都被分成了多个不同的模块，所以根据共享内存的模块组织方式可以把多处理机系统分成 3 类，分别是一致性内存访问计算机(Uniform Memory Access，UMA)、非一致性内存访问计算机(NonUniform Memory Access，NUMA)和只高速缓存访问计算机(Cache Only Memory Access，COMA)。

MIMD 计算机的另一个大类是多计算机系统，多计算机系统在体系结构层没有共享的第一级内存。换句话说，在多计算机系统中，CPU 上运行的操作系统不能通过 Load/Store 指令访问其他计算机的内存，它只能通过 Send/Receive 这样的操作系统原语显式地发送消息并等待响应的方式和其他的 CPU 通信。操作系统具有通过执行 Load/Store 指令访问远程的内存的能力是多处理机系统不同于多计算机系统的最重要的特征。

多计算机系统又可以粗略地分成两大类。第一类是大规模并行处理机(Massively Parallel Processors，MPP)，这是一种价格昂贵的超级计算机，它是由许多 CPU 通过专用的高速互联网络紧密耦合在一起组成的。第二类多计算机系统是由普通的 PC 或者工作站组成的，它们可能被放置在一个大的机架上，相互之间通过商用的网络连接起来，比较常用的有工作站网络(Network Of Workstations，NOW)和工作站集群(Cluster Of Workstations，COW)。

在本章后面部分，我们还将对 SIMD 阵列处理机、SIMD 向量处理机、MIMD 多处理机和 MIMD 多计算机系统进行进一步的讨论。

13.2　并行计算机系统的设计问题

13.2.1　并行计算机系统的互联网络

并行计算机的通信体系结构是系统的核心，它由两部分组成：底层的互联网络和上层的语言、软件工具包、编译器、操作系统等提供的通信支持。底层的互联网络是并行计算机系统内部的互联网络，它一般由以下 5 个部分组成：CPU、内存模块、接口、链路和交换结点。CPU 和内存的内容在前面的章节中已经讨论过，这里也就不再赘述。一般来说，它们是通信的端点。

接口是从 CPU 和内存取得消息并向另外的 CPU 和内存发送消息的设备。在许多设计方案中，接口是一块芯片或者是插在 CPU 局部总线上的一块电路板，这样就可以和 CPU 以及 CPU 的本地内存进行交互(如果有本地内存)。接口内部一般都有一个可编程的处理器，以及一些控制逻辑和控制存储器。此外还有一些私有的 RAM，这些 RAM 一般作为输

入和输出缓冲器。通常接口都具有读写不同内存的能力,因为这样才能够移动数据块。

链路是传送数据位的物理信道。链路可以是电缆、双绞线或者光纤,可以是串行的(1位宽)也可以是并行的(多于1位)。每种链路都有其最大带宽,也就是链路每秒能够传送的最大比特数。链路可以是单工的(单方向传送)、半双工的(某个时刻只能传送一个方向的数据)和全双工的(同时两个方向传送)。链路使用的时钟机制可以是同步或是异步的。

交换结点是互联网络的信息交换和控制站点,它是具有多个输入端口和多个输出端口的设备。当一个分组到达交换结点的某个输入端口时,交换结点将使用分组中的某些位来选择分组的输出端口。通常,交换结点的每个输入端口有接收器和输入缓冲器,每个输出端口有输出缓冲器和发送器。此外还有实施路径选择的控制逻辑等。

在设计和分析互联网络时,有几个问题是很重要的。首先就是互联网络的拓扑结构(Topology);第二个问题就是互联网络的交换结点如何工作;第三个问题是为了能把消息高效地传递到目的地应该使用何种寻径算法。下面就简要地讨论一下这3个问题。

1. 互联网络的拓扑结构

互联网络的拓扑结构描述了链路和交换结点是如何组织安排的。拓扑结构可以用图来表示,链路用边表示,交换结点用结点表示,如图13.4所示。互联网络中的每个结点都有边与之相连。数学上把和某个结点相连的边的数量称为结点的度(Degree)。一般来说,结点的度越大,寻径选择能力就越强,容错能力也越强。容错的意思是说当某条链路失效时可以绕过这条线路继续保持系统正常工作。如果网络中所有结点的度都相同,此网络称为对称网络,否则称为非对称网络。

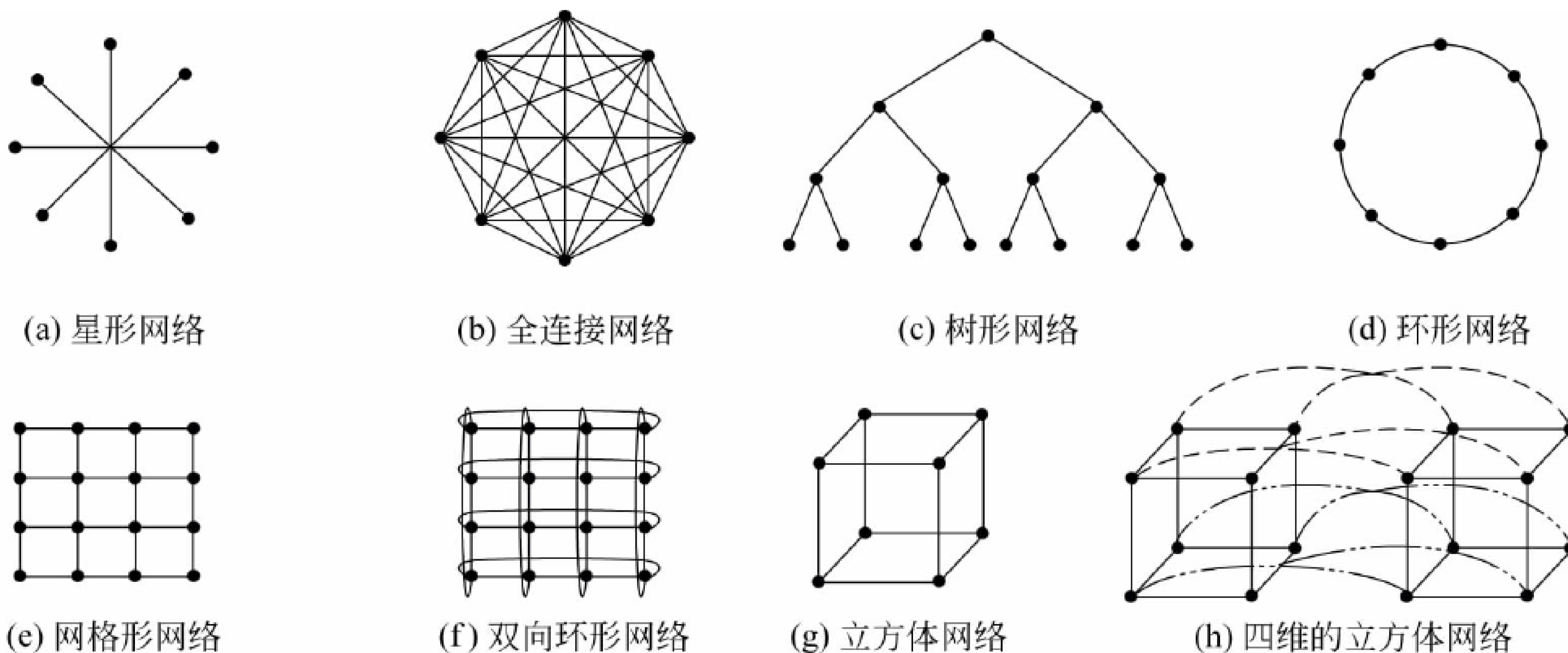


图13.4 互联网络的拓扑结构

互联网络的另一个属性是直径。如果使用两个结点之间的边数来表示两个结点之间的距离,那么图的直径就是图中相距最远的两个结点之间的距离。互联网络的直径直接关系到CPU和CPU之间以及CPU和内存之间交换分组时的最大延迟,因为通过每条链路都要花费一定的时间。直径越小,最坏情况下的性能就越好。两个结点之间的平均距离也很重要,因为它关系到分组的平均传递时间。

传输能力是互联网络的另一个很重要的特性,也就是每秒能传递多少数据。互联网络的传输能力的指标之一被称为对分带宽(Bisection Bandwidth)。为了计算对分带宽,首先需要通过移走图中的一些边,把网络分成两个结点数相等而且不相连的部分。然后计算移

走的边的带宽之和。因为可以有许多种把网络划分成两个相等部分的方式，因而也就会得到许多个值，对分带宽就是所有这些值中最小的。这个数值的意义在于如果对分带宽是800b/s，而且网络的两部分之间有大量的通信，那么整个流量就会被限制在800b/s之内。许多设计者认为对分带宽是互联网络最重要的性能指标。

图13.4中列出了几种拓扑结构。图中只画出了链路（边）和交换结点（点）。内存和CPU没有画出来，它们是通过接口连接在交换结点上的。图13.4(a)是一个星型(Star)网络，CPU和内存连接在外围的结点上，中间结点只做交换。虽然这种设计很简单，但是对一个大系统来说，中间的交换结点可能成为一个主要的瓶颈。另外，从容错的角度来看，这也不是一个好的设计，因为如果中间的交换结点出了问题整个系统就将崩溃。

图13.4(b)是全连接(Full Interconnect)网络。每个结点和任何一个其他的结点之间都有一条边。这种设计的对分带宽最大，直径最小，而且容错性能极好（损失任意6条边仍然能够完全相连）。但是不幸的是，对于 k 个结点来说，全连接需要 $k(k-1)/2$ 条边，当 k 比较大时，边的数量会太大以至于不可能实现。

图13.4(c)是树(Tree)型网络。这种设计的问题是对分带宽等于单条链路的容量。一般来说，靠近树的顶部的结点流量比较大，因此顶部几个结点将成为瓶颈。解决这一问题的一种方法是给顶部的链路增加带宽来增大对分带宽。例如，如果最底层的链路的容量是 b，上一层的容量就是 $2b$，顶层链路的容量就是 $4b$。这种设计方案称为胖树(Fat Tree)，该方案已经用于某些商用多计算机系统了。

图13.4(d)是环型(Ring)网络，而图13.4(e)是网格型(Grid或者Mesh)网络，许多商用系统都使用了这种结构。这种设计很有规律，易于扩展，而且直径与结点数的平方根成正比。网格型网络的一种变体是图13.4(f)中所示的双向环型(Double Torus)网络，这是一种把边缘结点连接起来的网格型网络。它不仅容错性能高于网格型网络，而且直径也比网格型网络小，因为对角的结点之间只有两条边。

图13.4(g)中的立方体(Cube)网络，是一种规则的三维拓扑结构。这里画的是一个2×2×2的立方体，一般情况下可以是 $k\times k\times k$ 的立方体。图13.4(h)中是一个四维的立方体，它是通过把两个三维立方体相应的结点连接起来而组成的。n 维的立方体称为超立方体(Hypercube)，许多并行计算机使用这种拓扑结构是由于直径随着维数线性增长，以及直径是结点数以2为底的对数。但是超立方体获得比较小的直径是以结点度的增加作为代价的，也就是链路的数量很大。虽然如此，超立方体仍然是高性能系统的通常选择的方案。

总结上面的讨论，假设互联网络规模即结点数目为 N，可以得到如表13.2所列的各种互联网络的特性列表。

表13.2 互联网络特征列表

网络拓扑	结点的度	网络直径	对分带宽	网络维数	备注
星形	$N-1,1$	2	$N/2$	0	
环形	2	$N/2$	2	1	
全连接	$N-1$	1	$(N/2)^2$	0	
树形	3	$2(k-1)$	1	0	$k=\log_2 N$，树的层数

续表

网络拓扑	结点的度	网络直径	对分带宽	网络维数	备　注
网格形	4,3,2	$2(n-1)$	n	2	$n=\sqrt{N}$
双向环形	4	n	$2n$	2	$n=\sqrt{N}$
立方体	k	k	N/2	k	$k=\log_2 N$,网的维数

除了上述的几种拓扑结构之外还有多种其他类型的网络结构,这里就不逐一介绍了。前面介绍的各种互联网络中的连接方式在系统的运行过程中不会改变,从这个意义上讲它们都是静态互联网络。与此相对的就是动态互联网络,动态互联网络以交换通道来实现,包括总线、交叉开关和多级交换网络等,动态互联网络常用于 UMA 和 NC-NUMA 多处理机中,我们将在共享内存的多处理机一节中对动态互联网络进行进一步的讨论。

2. 互联网络的交换结点

互联网络是由交换结点和连接它们的链路组成的。图 13.5 是一个小的由 4 个交换结点组成的网络。每个交换结点都有 4 个输入端口和 4 个输出端口。另外,每个交换结点都有 CPU 和互联电路(在图中并没有全部画出来)。交换结点所做的工作就是接收到达输入端口的分组然后把分组发送到正确的输出端口。每个输出端口都通过串行和并行链路连接到另一个交换结点的输入端口,这些链路在图 13.5 中用虚线表示。

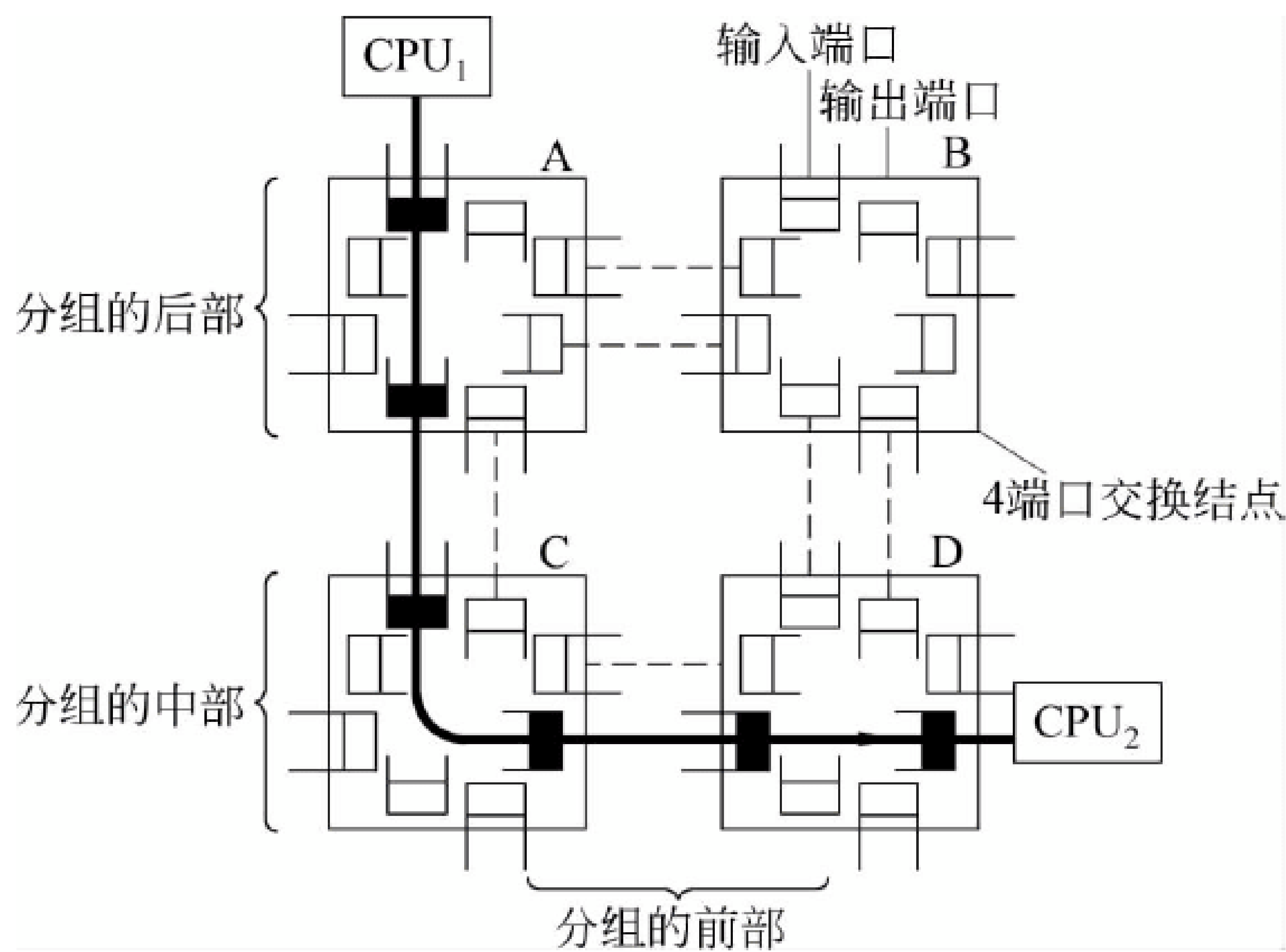


图 13.5　4 个交换结点的方型互联网络

交换结点有多种设计方案。一种方案称为电路交换(Circuit Switching),在发送分组之前,需要预先确定从源到目的地的整个路径。所有的端口和缓冲区都预先分配好,这样当传输开始时,所有必须的资源都确保是可用的,数据就可以全速地从源结点通过交换结点到达目的结点。图 13.5 中的交换结点就使用了电路交换,从 CPU_1 到 CPU_2 的粗实线就是一条预定的电路。这条电路使用了 3 个输入端口和 3 个输出端口。

对于电路交换来说,延迟时间是电路建立时间和传输时间之和。为了建立一条电路,首先要发送一个探测分组预约资源并报告结果。在发送探测分组时,可以同时装配数据分组。当电路建立以后,数据分组就可以全速传送。

电路交换的优点是,分组发送时能无竞争无干扰地全速传送。其缺点是需要提前预定

和预留，网络资源使用效率低。另外，为建立通路常常是先发送一个小的试探分组去预留资源，然后返回一个报告，这种通路建立时间的开销是比较大的。

第二种交换方案是存储转发分组交换(Store-and-forward Packet Switching)。这种方案不需要事先预约资源。源结点把整个分组发送给第一个交换结点，第一个交换结点把分组完全保存在自己内部。在图13.6(a)中，CPU_1是源结点，而目的地是CPU_2的整个分组，将首先被缓存在交换结点A中。一旦A获得了完整的分组，就把分组传递给交换结点C，如图13.6(b)所示。在整个分组到达交换结点C后，分组将被送往交换结点D，如图13.6(c)所示。最后，分组被送到目的地CPU_2。注意一下，这里不需要预先建立连接也不需要事先预定资源。

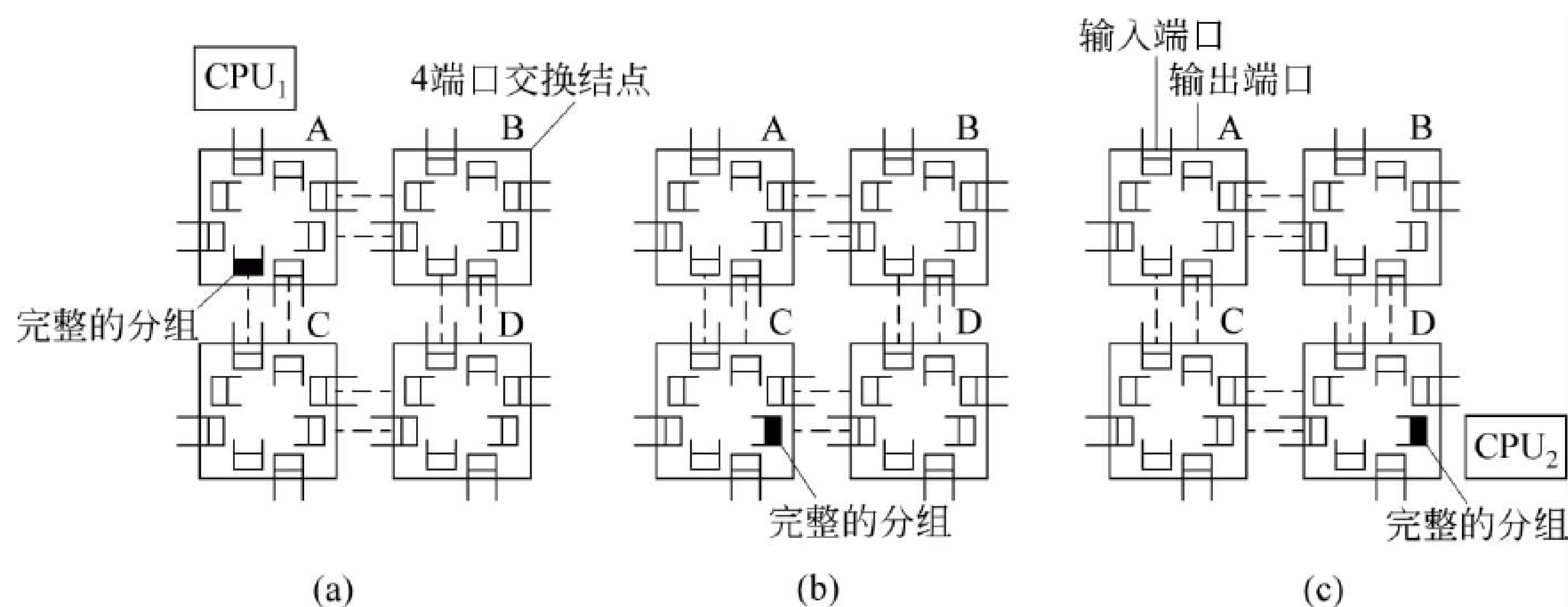


图13.6　存储转发分组交换

存储转发交换结点必须能够缓冲分组，因为当数据源(比如，CPU、内存和交换结点)需要传送分组时，需要的输出端口可能正忙于传送另外一个分组。如果没有缓冲，当一个正在到达的分组需要使用一个正在被占用的端口的话，该分组就不得不被丢弃，势必导致互联网络的可靠性下降。常用的缓存策略有输入缓存、输出缓存和公共缓冲区3种。

存储转发分组交换的优点是不需要预约资源，灵活而有效；缺点是它增大了分组通过互联网络的延时。而且对分组交换网络来说，虽然不需要事先向目的结点发送探测分组，但是仍然需要装配分组的时间。此外，为保存最大的分组和避免几条通路向同一个结点传送时造成分组的丢失，分组缓冲区应比较大，这不利于VLSI的实现。

解决该存储转发分组交换延时较长问题的一种方案是设计一种同时具有电路交换的特点和分组交换的特点的混合网络。例如，可以从逻辑上把分组分成小的单元。只要第一个单元到达了交换结点，就可以被传送到下一个交换结点而不需要等待所有的单元都到达交换结点。这种策略不同于电路交换，因为它不需要事先进行端到端的资源预约。因此可能出现资源(端口和缓冲区)的竞争。在虚拟直通寻径(Virtual Cut Through Routing)策略中，当分组的第一个单元不能移动时，分组的其余单元可以继续向第一个单元所在的结点传送，因此在最坏的情况下，虚拟直通就变成了存储转发分组交换，它仍然需要较大的缓冲区。而且这种情况下的延时与存储转发分组交换方式一样，这也是虚拟直通寻径方式的一个缺点。

虚拟直通寻径是对存储转发分组交换方式的改进，虫蚀寻径(Wormhole Routing)又是对虚拟直通寻径方式的改进。在虫蚀寻径策略中，当第一个单元不能移动时，通知源结点，

源结点就停止传送,因此分组就像一条虫子一样停留在两个或者更多个交换结点中。当需要的资源可用后,分组继续前进。这种寻径方式的明显优点是,各个结点不再需要大的分组缓冲区,只要较小的分组单元缓冲区就可以了,这样也有利于 VLSI 实现。它的缺点是,头单元被阻塞时后面所有的数据单元也被阻塞,占用了较多的结点(相比之下,虚拟直通方式头单元受阻时只占用一个结点)。

虚拟直通和虫蚀寻径不需要发送探测分组建立电路,也没有存储转发延时。因此,一般来说,延迟时间就是初始装配分组的时间加上发送分组的时间,当然,在所有的延迟时间中都应该有传送延迟,但是一般来说,传送延迟很小。

3. 互联网络的寻径算法

在任何一个维数大于 1 的网络中,都需要选择通过哪条路径从源结点到达目的结点。一般情况下,存在多条路径。决定一个分组从源结点到达目的结点的过程中经过的结点序列的算法称为寻径算法(Routing Algorithms)。

在网络中一般都存在多条路径,因此需要好的寻径算法。一个好的寻径算法能够在多个链路上分担负载来充分利用带宽。另外,寻径算法还必须避免在互联网络中出现死锁(Deadlock)现象。当互联网络中正在传送的多个分组都占用着别的分组需要的资源将导致死锁,即所有的分组都不能继续传送而进入无限等待资源的情况。

寻径算法可以分成源寻径(Source Routing)和分布式寻径(Distributed Routing)两大类。在源寻径中,源结点预先决定穿过互联网络的完整的路径,使用路径中每个结点的端口号的列表来表示。在分布式寻径算法中,每个交换结点自己决定把到达的分组发送到哪个输出端口。一般来说,在各个交换结点都设立一个路径表,而分组的头部含有一个寻径字段,用于说明分组的目的地址和选择路径的依据。

如果算法对所有到相同目的结点的分组都做出相同的决策,那么这样的寻径算法就称为静态的(Static);如果算法在做路径选择时考虑了当前情况,该算法就是自适应的(Adaptive),自适应算法要比静态算法复杂得多。有关这些内容的深入讨论请读者参考计算机网络类书籍和相关资料。

13.2.2 并行计算机系统的性能问题

设计并行计算机的目的就是使它的运行速度比单处理器的计算机快。如果不能实现这一目标,那么所有的努力都是徒劳的。此外,我们还应该用尽可能高效率的方式实现这一目标。一台比单处理器的计算机快两倍但是价格却贵 50 倍的并行计算机肯定是无人问津的。下面将从并行计算机系统的主要硬件和软件性能指标以及如何获得更高的性能等方面来讨论并行计算机体系结构的性能问题。

1. 硬件性能指标

从硬件的角度来说,重要的性能指标是 CPU 和输入/输出的速度以及互联网络的性能。CPU 和输入/输出的速度和单处理器的情况一样,因此并行计算机中关键的硬件性能指标就是互联网络的性能。互联网络的性能有两个重要的指标:延时(Latency)和带宽(Bandwidth)。

延迟时间是指从 CPU 发送分组至接收到响应的时间间隔。如果分组是发送到内存去的,那么延迟时间就是读写一个内存字或者一块内存区的时间。如果分组是发送到另一个

CPU的,延迟时间就反映了使用该大小分组的处理器之间的通信时间。延迟时间由多个因素决定,而且正如我们在上一节中分析的那样,电路交换、分组交换、虚拟直通路和虫蚀寻径的延迟时间都是不同的。

另一个硬件性能指标是带宽。许多并行程序,特别是用于科学计算的并行程序往往需要移动大量的数据,因此系统每秒能够移动的比特数就成了系统比较关键的性能指标。关于带宽有多个性能指标。我们前面已经讨论了对分带宽,另一个带宽指标是聚集带宽(Aggregate Bandwidth),它是把所有链路的带宽加在一起而得到的。聚集带宽给出了系统能够同时传送的最大的比特数。此外,还有一个重要的带宽指标是按照CPU能力计算的平均带宽(Average Bandwidth)。

2. 软件性能指标

从软件的角度来看,最关键的性能指标是加速比。一个程序在有 n 个处理器的计算机上运行和在只有一个处理器的计算机上运行相比要快多少倍。

一些典型的结果如图13.7所示。图中画出了在由64个Pentium Pro CPU组成的多计算机系统上运行几个不同的并行程序的结果。每条曲线都反映了一个程序的加速比,加速比是CPU的数量 k 的函数。最理想情况下的加速比如图13.7中的虚线所示,使用 k 个CPU将使程序运行快 k 倍,并且对于任意的 k 都成立。很少有程序能够获得理想的加速比,但是有些程序比较接近理想的加速比。

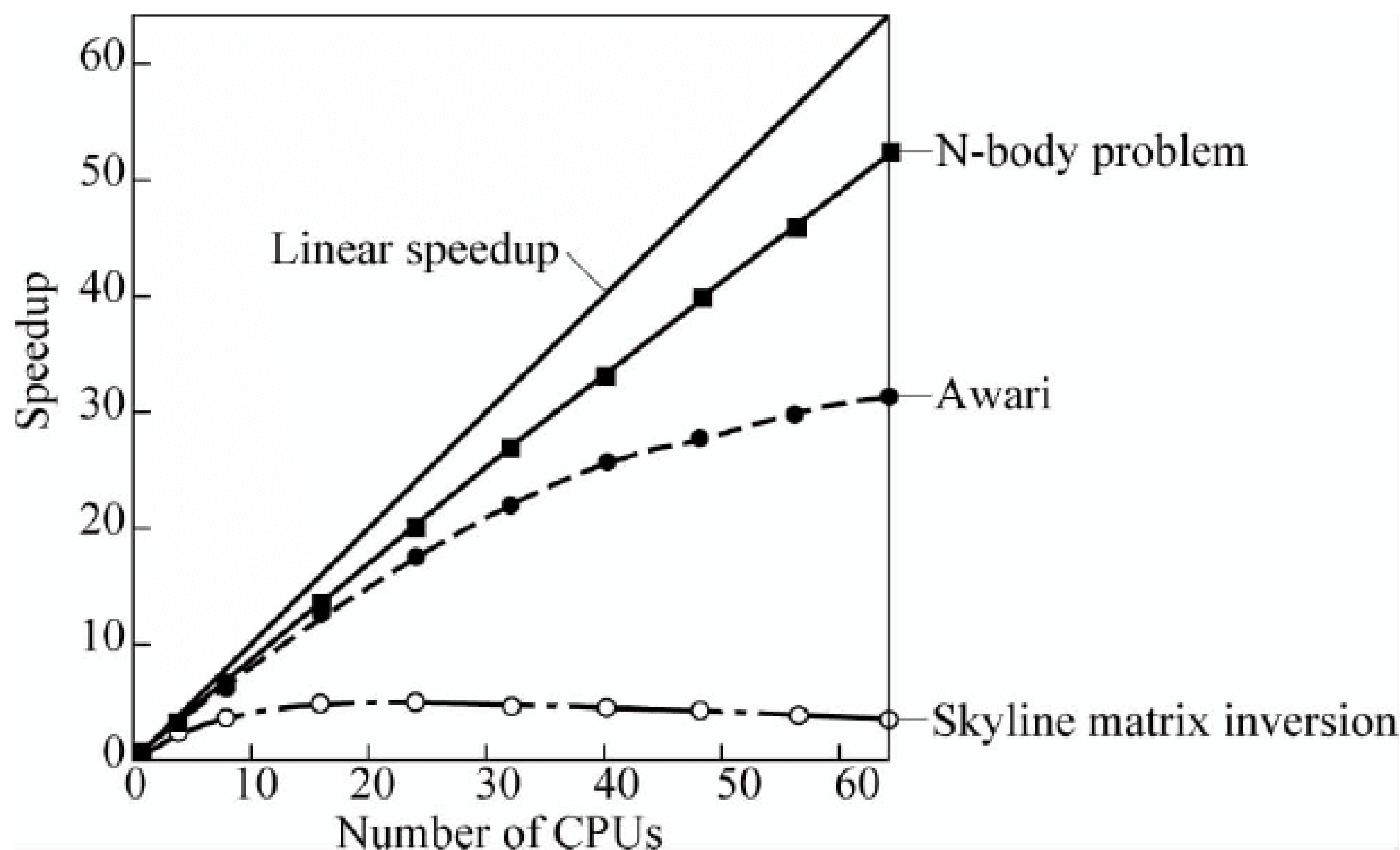


图13.7 实际程序获得的加速比总是低于线性加速比

理想的加速比不可能达到的部分原因在于几乎所有的程序都有串行部分,比如程序的初始化、数据读入和结果合并等。在这些地方,CPU再多也没有用。假定一个程序在单处理器的计算机上运行需要 T 秒,其中一部分是串行代码,所占比例记为 f,那么剩余的 $(1-f)$ 就是可以并行的。如果后一部分代码运行在 n 个CPU上而且没有任何其他开销,那么在最理想的情况下,执行时间可以从 $(1-f)T$ 减少到 $(1-f)T/n$。那么串行部分加并行部分的整个执行时间就是 $f\times T+(1-f)T/n$。加速比就是原来程序的执行时间除以新的程序的执行时间,即

$$\text{Speedup}=\frac{n}{1+(n-1)f}$$

如果 $f=0$,我们就可以获得线性加速比;但是如果 $f>0$,就不可能得到理想加速比,因为存在串行部分。这就是Amdahl定律。但是,Amdahl定律并不是不能获得理想加速比的

唯一原因。通信延迟时间、有限的通信带宽和算法的效率都会影响程序的加速比。

3. 获得更高的性能

提高性能的最直接的办法就是给系统增加更多的CPU。但是,增加CPU时要注意不要产生任何瓶颈。为了更好地理解前面所提到的可扩展性的含义,我们来看一下图13.8(a)中的4个CPU通过总线相连的例子。现在我们给该系统增加12个CPU,使CPU数目增加到16个,如图13.8(b)所示。如果总线的带宽是bM b/s,那么由于CPU数量增加了4倍,每个CPU可用带宽就从$b/4$ M b/s降低到b/16 M b/s。这样的系统被认为是不可扩展的。

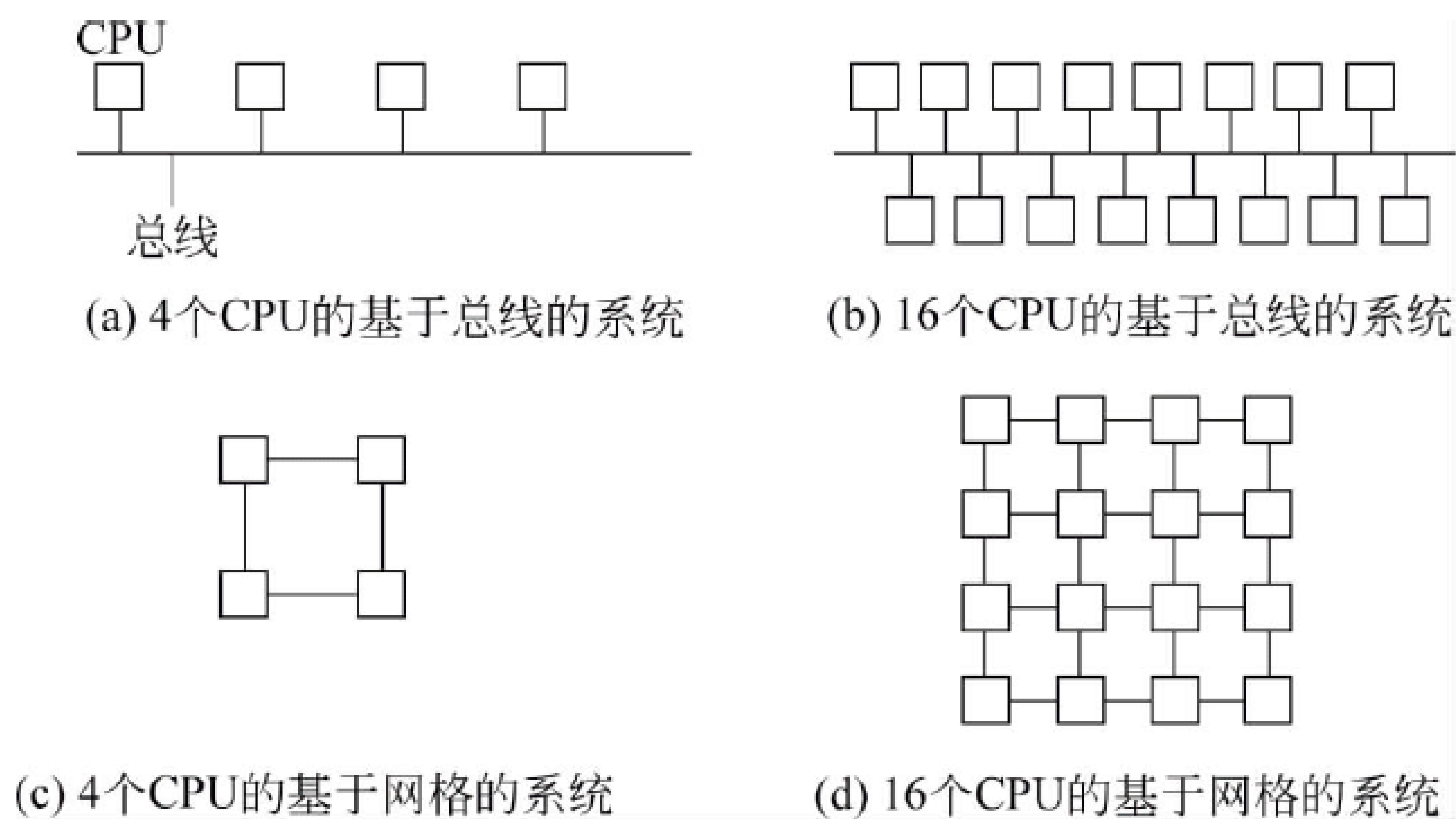


图13.8 不同数目CPU的互联网络

下面对基于网格网络的系统做同样的扩展,如图13.8(c)和13.8(d)所示。使用这种拓扑结构,增加新的CPU时也要增加相应的链路,因此系统的扩展并不会像基于总线的系统那样导致每个CPU的平均带宽的下降。实际上,链路和CPU的比例从4个CPU时的1.0(4个CPU,4条链路)提高到了16个CPU时的1.5(16个CPU,24条链路),因此,CPU的增加相应地提高了CPU的平均带宽。

当然,带宽并不是唯一的问题。为总线系统增加CPU并不增加互联网络的直径以及没有竞争情况下的线路延时,而网格网络系统则不是这样。对于$n \times n$的网格网络来说,直径是$2(n-1)$,因此最坏情况的延时增长大约是和CPU数目的平方根成正比的。

理想情况下,一个可扩展的系统随着CPU的增加应该能够保持相同的CPU平均带宽和不变的平均延迟时间。而在实际中,保持CPU足够的带宽是可以做到的,但是在所有实际的设计方案中,延迟时间总是随着CPU数量的增长而增长。使延迟时间按照CPU数量的对数增长,就像超立方体那样,就已经是最好的方案了。

13.2.3 并行计算机系统的软件问题

软件问题在讨论并行计算机体系结构的时候也是非常重要的,如果没有并行软件,并行硬件基本上就没有什么用处,因此,好的硬件设计师在设计硬件时就考虑了软件的需求。并行计算机的软件有4种一般的设计方法。

第一种方法是为普通的串行语言增加特殊的函数库。例如,程序员可以编写调用转换大矩阵的库函数和解偏微分方程组的库函数的串行程序而不用知道这些库函数实际上是并行执行的。这种方法的问题在于只是在很少的库函数中实现了并行性,而大量的程序代码

仍然是串行的。

第二种方法是为编程语言增加包括通信和控制原语的库函数。程序员仍然使用传统的编程语言编程，但是程序员需要使用这些原语来创建和管理并行性。

第三种方法是为现有的编程语言增加一些特殊的结构，比如可以很容易地创建新的并行进程的能力，并行执行循环的能力和同时对一个向量的所有元素执行算术运算的能力。这种方法目前被广泛使用，许多编程语言都被修改成包括这些并行机制。

第四种方法是发明一种全新的用于并行处理的语言。使用新语言的一个显而易见的优势是这样的语言肯定很适合于并行处理，但是缺点也同样明显，程序员必须学习一种新语言。

虽然目前已经有许多用于并行编程的函数库、编程语言扩展和新的并行编程语言，但是所有并行计算机软件的核心不外乎5个关键问题，即控制模式、并行粒度、计算模式、通信方式和同步原语，一个好的并行计算机软件必须解决好这几个问题。

13.3 SIMD 计算机简介

SIMD 是单指令流多数据流计算机，主要用于解决使用向量和阵列这样比较规整的数据结构的复杂的科学计算和工程计算问题。这种计算机只有一个控制单元，每次只能执行一条指令，但是这一条指令可以同时对多个数据进行操作。SIMD 计算机可以分为阵列处理机和向量处理机两大类。下面我们就对这两种类型的计算机进行简单的介绍。

13.3.1 阵列处理机

阵列处理机的思想早在40多年前就提出来了。然而，在那以后又过了10年第一台为NASA 服务的阵列处理机 ILLIAC Ⅳ才建造完成并投入实际使用，ILLIAC Ⅳ也是 SIMD 计算机的原型。从那时起，多家公司都开始制造商用的阵列处理机，包括 Thinking Machine 公司的 CM-2 和 Maspar 公司的 MP-2，但是没有一家公司在市场上取得较大的成功。

阵列处理机使用的基本思想是一个单一的控制单元提供信号驱动多个处理单元同时运行，如图13.9所示。每个处理器单元都由 CPU 或者是功能增强的 ALU 和本地内存组成。由于所有的处理单元都是由一个控制单元驱动的，因此它们的执行是同步的。虽然所有的

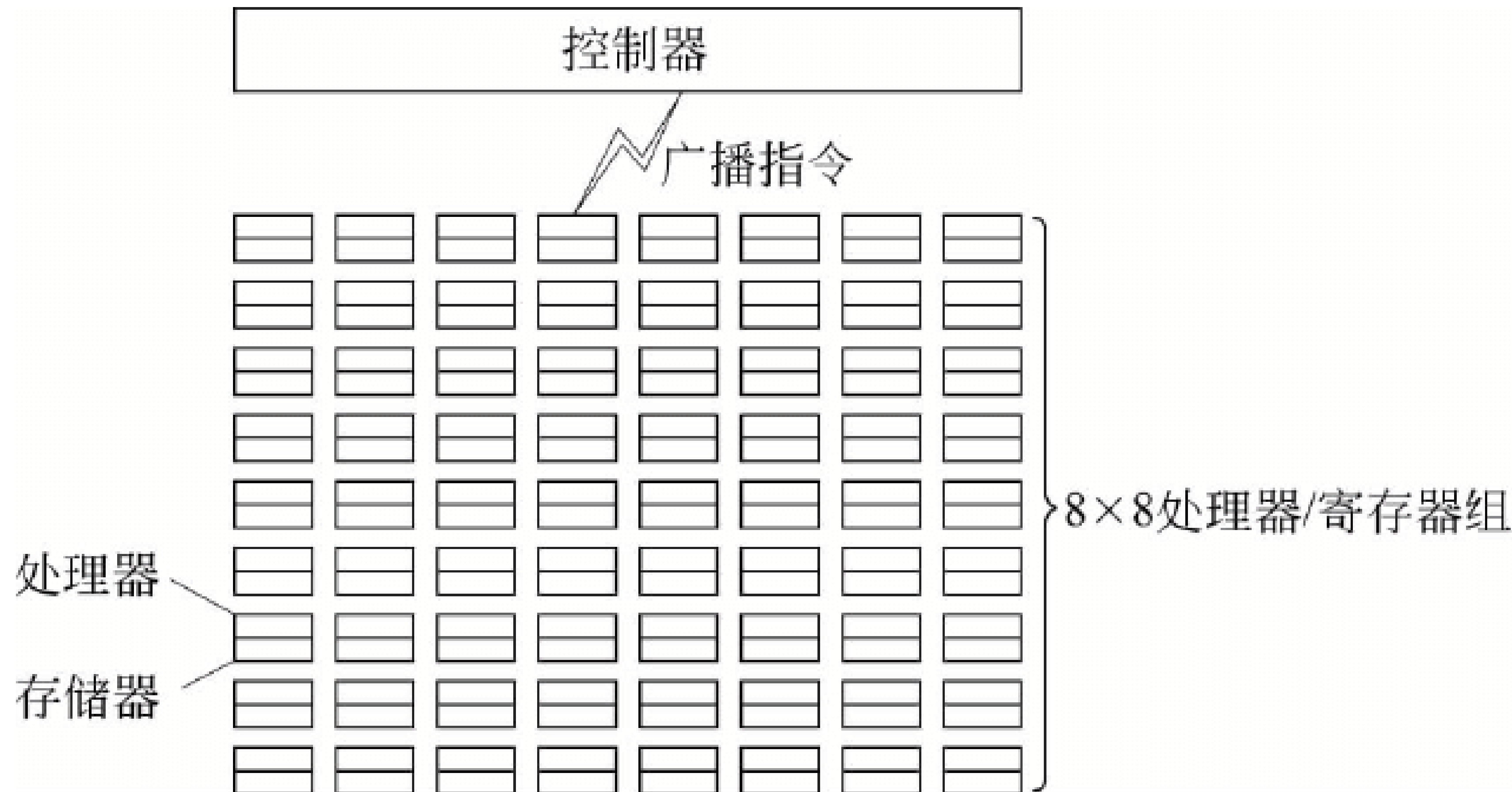


图 13.9　ILLIAC Ⅳ型阵列处理机

阵列处理机都遵循这一通用的模式，但是在具体设计时不同的阵列处理机仍然有不同之处。

第一个不同就是处理单元的结构。处理单元的结构可能很简单，也可能很复杂。从简单的角度来说，最简单的处理单元可能就是一个1位的ALU，就像CM-2中的处理单元那样。处理单元也可以采用8位的ALU、32位的ALU或者带浮点计算能力的更加强劲的处理单元。从某种程度上来说，采用什么样的处理单元取决于机器的设计目标。如果是用于数值计算，就需要浮点计算能力；如果是用于信息处理，也可能不需要浮点计算能力。

第二个不同是处理单元如何连接。从原理上来说，图13.4中列出的所有的拓扑结构都是可行的。方格型网络是比较常用的结构，它们适合处理许多二维的问题包括矩阵计算和图像处理，方格型网络还具有良好的可扩展性，在增加处理器数量的同时可以自动地增加带宽。

第三个不同是处理单元的自治能力有多强。在我们讨论的这种设计中，控制单元告诉处理单元执行什么指令，但是在许多阵列处理机中，每个处理单元都可以选择执行或者不执行某条指令，选择的依据是处理单元的本地数据，比如条件代码中的某些位。这一特性给阵列处理机带来了相当大的灵活性。

从并行计算机系统体系结构来看，阵列处理机没有太好的发展前景，这里就不再对它进行更详细的介绍。

13.3.2 向量处理机

另一种类型的SIMD计算机是向量处理机，向量处理机在商业上取得了很大的成功。Cray Research公司设计的系列计算机，从1976年的Cray-1到后来的C90和T90，在科学计算领域占据了数十年的统治地位。下面将介绍向量处理方式和向量处理机的基本原理。

典型的数值计算的应用，可以由类似下面这样的语句组成：

```
for(i=0;i<N;i++)
    A[i]=B[i]+C[i]
```

这里的A、B和C通常是浮点数组成的数组。该循环的功能把数组B和C的第i个元素相加并把结果保存在数组A的第i个元素中。实际上我们可以把数组A、B、C看作是长度为N的向量(Vectors)，即

$$A=(a_1a_2\cdots a_N)B=(b_1b_2\cdots b_N)C=(c_1c_2\cdots c_N)$$

引入向量数据表示之后，上述循环语句可以写成如下向量运算方式：$A=B+C$。向量运算可以采用3种不同处理方法：横向处理方法、纵向处理方法和纵横处理方法。现以计算表达式$D=A\times(B+C)$为例进行说明，其中A、B、C、D都是长度为N的向量。

横向处理方法：向量计算按行的方式从左至右横向进行。

$$d_1=a_1\times(b_1+c_1)$$
$$d2=a_2\times(b_2+c_2)$$
$$\vdots$$
$$d_N=a_N\times(b_N+c_N)$$

即逐个求向量D中N个分量：先进行相加运算$k_1\leftarrow(b_1+c_1)$，k_1为暂存单元；然后进行相乘运算$d_1\leftarrow k_1\times a_1$。可以看出，当采用流水方式计算时，在每个向量加乘运算中都会发生

数据相关。而且当使用静态流水线时，还要进行2次乘和加功能的转移。这样共出现 N 次相关和 $2N$ 次功能转换。因此，横向处理方法不适合于向量流水处理。

纵向处理方法：向量计算是按列的方式自上而下纵向地进行。

$$d_1 = a_1 \times (b_1 + c_1) \rightarrow k_1$$
$$d_2 = a_2 \times (b_2 + c_2) \rightarrow k_2$$
$$\vdots$$
$$d_N = a_N \times (b_N + c_N) \rightarrow k_N$$

即先是所有 B 和 C 向量元素对的相加运算，中间结果暂存到 $k_1 \sim k_N$ 中；然后再纵向加工所有对应元素的乘法运算。用向量指令形式来表示，则变成 $K=B+C$，$D=K\times A$。显然，当采用流水方式计算时，数据相关在两条向量指令间仅有1次，而流水线加、乘功能的切换只需1次。因此，纵向处理方法可获得较高的吞吐率，适合于在向量处理机中应用。例如存储器-存储器工作方式的向量处理机都采用纵向处理方法。

纵横处理方法：纵横处理方法是上述两种方法的结合，又称为分组处理方法。组内采用纵向处理，组间采用横向处理。假设向量长度为 N，分成 S 组，每组长度为 n；r 为余数，也作为一组处理，共 $S+1$ 组。即 $N=Sn+r$，其中，$n\leqslant N$，$r<n$，所有参数均为正整数。

先算第一组：$k_{1\sim n}=b_{1\sim n}+c_{1\sim n}$

$d_{1\sim n}=a_{1\sim n}\times k_{1\sim n}$

再算第二组：$k_{n+1\sim 2n}=b_{n+1\sim 2n}+c_{n+1\sim 2n}$

$d_{n+1\sim 2n}=a_{n+1\sim 2n}\times k_{n+1\sim 2n}$

再继续第三组，组内仍采用纵向处理。最后一组（$S+1$）为余数 r，因 $r<n$，仍作为一组来处理。由上可见，每组内各有两条向量指令。各组内有一次数据相关，需2次流水功能切换，且需 n 个中间向量暂存单元。纵横处理方法适合于寄存器-寄存器工作方式的向量处理机。

图13.10是一种可行的适用于向量处理的SIMD体系结构。通过执行向量运算指令，这种计算机输入两个具有 N 个元素的向量，可以同时对 N 个元素进行操作的向量ALU对相应的元素进行并行处理，最后产生一个结果向量。输入向量和输出向量可以保存在内存中，也可以保存在特殊的向量寄存器里。

在实际使用中，很少有超级计算机采用图13.10的体系结构，常用的方法是把向量处理和流水线结合起来，这样能够获得较好的性能价格比。需要注意的是，用于执行向量操作的流水线和执行通用指令的流水线有一个很大的不同，在执行向量操作时不会遇到分支跳转指令。也就是几乎不会发生数据相关和控制相关问题，这样每个时钟周期都可以充分利用。

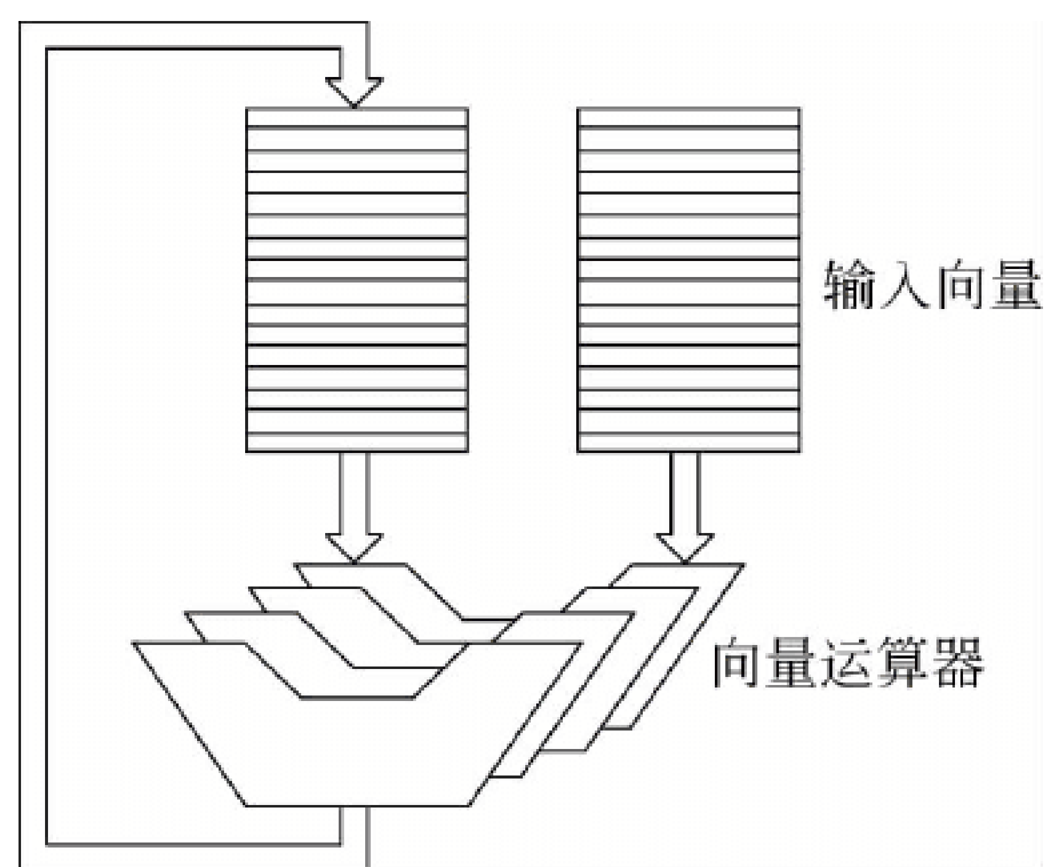


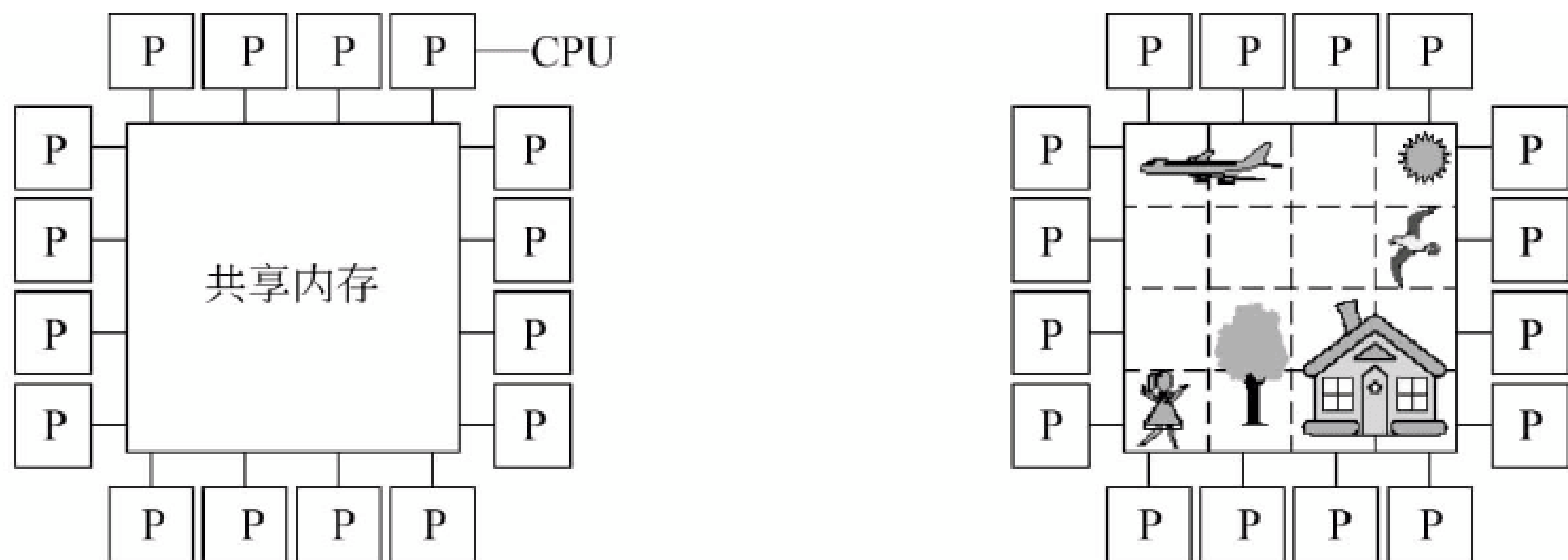
图13.10　用于向量处理的SIMD体系结构

向量处理机一般具有多个ALU，每个都专门处理某种特殊类型的操作，所有的ALU都可以并行执行。典型的例子就是早期的向量超级计算机Cray-1，它类似于RISC的体系结构使它成了一个很好的研究范例，许多现代的向量超级计算机的体系结构都受到它的影响。

虽然向量流水处理具有很多结构上的特点,但是要使软件能够充分发挥硬件所提供的这些特点并不容易。它必然要对语言结构和编译程序提出新的要求,例如高级语言需要增加向量运算符等。另外,优化的目标程序必然要和向量流水机器的具体结构特点密切相关,这也将导致编译程序的设计复杂化。

13.4 共享内存的多处理机系统

多处理机系统是具有多个CPU,并且所有的CPU共享一个地址空间的计算机系统,如图13.11(a)所示。多处理机系统有时也被称为共享内存系统(Shared Memory System)。许多计算机厂商都提供多处理机系统,比如Sun Enterprise 10000、Sequent NUMA-Q、SGI Origin 2000和HP/Convex Exemplar等。



(a) 16个CPU共享一个公共内存的多处理机系统　(b) 一个图像分成16块，每块都由不同的CPU分析

图13.11　多处理机设计方案

从软件的角度来说,多处理机系统很容易扩展。多处理机系统中的所有处理器都共享同一个映射到共享物理内存上的虚拟地址空间。任何一个处理器都可以通过执行LOAD或者STORE指令从内存读一个字或者向内存写入一个字,不需要附加任何其他的操作。两个处理器之间可以通过很简单的方式进行通信,只要一个处理器把数据写入内存而另一个处理器从内存中把数据读出就可以了。

由于多处理机系统中,两个或者多个处理器之间可以通过读写内存进行通信,因此多处理机系统很流行。这是一种程序员很容易理解的模型而且可以用于解决大量的问题。例如,一个程序需要检测一幅BMP图像并列出其中所有的对象。如图13.11(b)所示,该图像被调入内存,16个CPU每个都运行一个单独的进程,每个进程负责分析图像的1/16。当然,每个进程都可以访问整个图像,这一点很重要,因为某些对象可能会占据图像的多个部分。如果某个进程发现某个对象延伸到了自己所处理的部分之外,那么它可以通过读相邻部分的图像来继续自己的分析。在这个例子中,某些对象可能会同时被多个进程发现,因此需要做一些协调工作来判断图中到底有多少房子、多少树和多少飞机。

和所有的计算机系统一样,多处理机系统也必须有磁盘、网络适配器和其他的输入/输出设备。在某些多处理机系统中,只有特定的几个CPU才能访问输入/输出设备,因此也就具有特殊的输入/输出函数。在其他的一些系统中,每个CPU都能平等地访问每个输入/输出设备。如果在一个系统中,每个CPU都能平等地访问所有的内存模块和输入/输出设备,而且在操作系统看来这些CPU是可以互换的,那么这种系统就是对称多处理机系

统(Symmetric MultiProcessor,SMP)。

多处理器系统中所有的CPU都共享公共内存,根据共享内存的实现方式可以把多处理器系统分成3类,分别是一致性内存访问计算机、非一致性内存访问计算机和基于Cache的内存访问计算机。之所以这样分类是因为在大多数多处理器系统中内存都被分成了多个不同的模块。UMA计算机的特点是CPU访问所有的内存模块的时间都相同。换句话说,读取每个内存字的时间是相等的。如果在实现中有困难,就把速度快的内存的访问速度降低以保证和最慢的相等,这样程序员就不会感觉到速度的差别了。这就是一致性的含义。这种一致性可以保证系统的性能可以预测,也有利于程序员编写高效率的代码。和UMA相反,在NUMA多处理器系统中就没有这种一致性。在NUMA系统中,靠近CPU的内存模块的访问速度比其他的内存模块快得多。这样实现也是出于提高性能的考虑,它主要关系到代码和数据的位置。COMA计算机也是不一致的,但是这两种不一致有所区别。后面我们会详细讨论这些类型和它们的子类型。

13.4.1 一致性内存访问的UMA多处理机系统

最简单的UMA多处理机系统是基于单总线的,如图13.12(a)所示。两个或者更多的CPU以及一个或者更多的内存模块都使用同一条总线进行通信。当某个CPU需要读取内存时,它首先检查总线是否正在被使用。如果此时总线是空闲的,CPU就把内存地址放在总线上,并且在其他一些控制信号的配合下,等待内存把它需要的内存字放在总线上。如果当CPU需要读写内存时发现总线正在被使用,那么它只有等待直到总线空闲。这种设计的主要问题是,虽然在使用两三个CPU情况下,对总线的争用还是可以管理的,但是如果使用32个或者更多个CPU,对总线的争用就是无法忍受的。系统的能力将受到总线带宽的限制,因为大多数CPU在大多数时间内都处于等待状态。

解决这一问题的一种方案是为每个CPU都增加Cache,如图13.12(b)所示。Cache可以在CPU芯片内部,CPU芯片的旁边或者处理器板上其他位置,也可以把这三者结合起来使用。由于许多读操作都可以从Cache中获得数据,总线的流量将会少得多,这样不但减轻了对总线的竞争,而且系统也可以支持更多的CPU。

在上述方案之外还有另一种可行的方案,如图13.12(c)所示,在这种方案中,每个CPU不仅有自己的Cache而且还有自己的私有内存,私有内存是通过私有总线进行访问的。为了能够最佳地利用这种体系结构,编译器应该把所有的程序文本、字符串、常量和其他的只读数据以及堆栈和局部变量等放在自己的私有内存中。共享内存只用于存放共享变量。在大多数情况下,这种精心考虑的数据分布可以极大地减少总线流量,但是它需要编译器的主动支持。

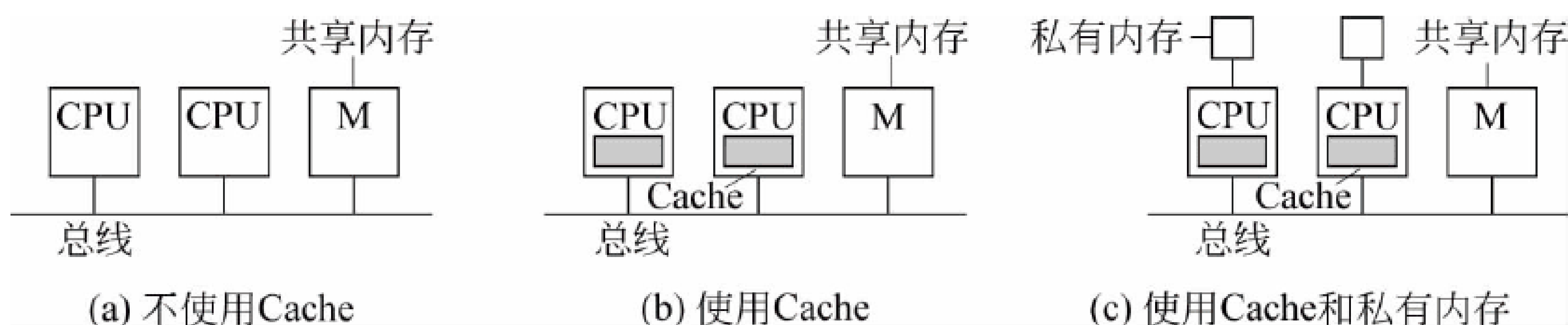


图13.12 基于总线结构的多处理机系统

我们在前面关于性能的讨论过程中,忽略了一个基本的问题。例如当 CPU_1 的 Cache 中有一行数据,而 CPU_2 同样也想读这个数据时,如果在没有任何特殊规定的情况下,CPU_2 的 Cache 中也将获得该数据的复制。从原理上来说,同一行数据缓存两次也是可以的。现在假定 CPU_1 修改了这一行数据,而就在修改刚刚完成之后,CPU_2 读取了它自己的 Cache 中的这行数据。显然,CPU_2 读到的是过时的数据,也就会导致 CPU_2 上运行的程序产生错误的结果。这一严重的问题称为 Cache 一致性问题。如果不想办法解决这个问题,Cache 就不能使用,基于总线的多处理机系统也就只能使用两三个 CPU 了。认识到问题的重要性后,人们提出了多种解决方案,也就是所谓的 Cache 一致性协议(Cache Coherence Protocols),虽然这些协议在细节上有所不同,但是它们的目的都是为了防止在两个或者更多的 Cache 中出现同一行数据的不同版本。

在几乎所有这些 Cache 一致性协议中,Cache 控制器都被设计成可以监听总线,可以监听其他 CPU 和 Cache 的所有请求并在某些特定的情况下采取相应的操作。正是因为它们可以监听总线,这种 Cache 被称为监听型 Cache。由 Cache、CPU 和内存共同实现的防止多个 Cache 中出现相同数据的不同版本的规则集合就组成了 Cache 一致性协议。Cache 读写和保存的单元称为 Cache 行,一般是 32 个字节或者 64 个字节。

1. 写直达 Cache 一致性协议

最简单的 Cache 一致性协议是写直达协议(Write Through Protocol)。表 13.3 中列出了监听型 Cache 按照此协议进行读写操作时的 4 种情况,通过此表可以很容易地理解该协议。表中的空白项表示不采取任何操作。

表 13.3　写直达 Cache 一致性协议

操作	本地请求	远程请求
读缺失	从内存取数据	
读命中	使用本地 Cache 的数据	
写缺失	修改内存中的数据	
写命中	修改 Cache 和内存	将 Cache 项置为失效

当 CPU 要读的字不在 Cache 中时,即发生了读缺失,Cache 控制器就把包括该字的一行数据读入 Cache。这行数据是由内存提供的,在该协议中,内存中的数据总是最新的。接下来的读操作就可以直接从 Cache 中获得数据,也就是读命中。

当发生写缺失时,也就是要写入的字不在 Cache 中,则把被修改的字写回内存,但是并不把包括该字的行调入 Cache。当发生写命中时,也就是要写入的字已经在 Cache 中,修改 Cache 的同时还要把该字直接写入内存。该协议的要点就在于所有的写操作都直接写入内存以保证内存中的数据总是最新的。

下面我们再从监听者的角度来看一看此协议中的这些操作,如表 13.3 右边一列所示。假设共有两个 Cache,Cache1 和监听的 Cache2。当 Cache1 读缺失时,它在总线上发送一个从内存中取数据的请求。Cache2 监听到该动作,但是没有做任何操作。当 Cache1 读命中时,总线上不会发送任何请求,因此 Cache2 也就没有监听到 Cache1 发生了读命中。

写操作就和读操作有所不同了,当 CPU1 执行写操作时,在写缺失和写命中两种情况

下 Cache1 都要在总线上发送写请求。无论是哪种写请求，Cache2 都要检查写入内存的字是否在自己的 Cache 中。如果不在，那么从 Cache2 的角度来看，这就是一个远程的写缺失请求因此不用做任何操作。需要注意的是，在表 13.3 中远程的写缺失意味着该字不在监听者的 Cache 中；它并不关心该字是否在写操作发起者的 Cache 中。因此某个请求可能对本地 Cache 来说是命中而对于监听者来说是缺失，反之亦然。

现在假定 Cache1 写入内存的字也存在于 Cache2 中，即发生了远程请求中的写命中。如果 Cache2 仍然不进行任何操作，那么相应的字将变成过时的数据，所以它需要在包括这个最新修改的字的 Cache 行上打上无效标记。因为所有的 Cache 都监视所有的总线请求，无论何时写入一个字，最后的结果都是操作发起者的 Cache 被更新，内存被更新，其他所有的 Cache 中的这个字都被设置为无效。通过这种方式有效地避免了不一致性的出现。

简单的方案往往效率很低，因为每次写操作都要通过总线，只要 CPU 的数量稍微多一些，总线就仍然成为瓶颈。为了保证总线的流量在一定范围之内，人们设计出了其他的 Cache 一致性协议。它们都具有一个共同的特点：写操作不直接写入内存。相反，当 Cache 行被修改后，Cache 中的某一位被设置以表示该 Cache 行中的数据是正确的而内存中的数据是过时的。当然最终该行将会被写回内存，但是可能是在多次写操作之后了。这种类型的协议被称为写回协议(Write-back Protocol)。

2. 写回 Cache 一致性协议

一种比较常用的写回 Cache 一致性协议是 MESI 协议，它是用协议中用到的 4 种状态的首字母(M,E,S 和 I)来命名的。它是从早期的写一次性协议(Write-once Protocol)发展而来的。目前 Pentium 和许多其他的 CPU 都使用了 MESI 协议来监听总线。在这个协议中每个 Cache 项都处于下面 4 种状态之一。

(1) 无效(Invalid)。该 Cache 项包含的数据无效。

(2) 共享(Shared)。多个 Cache 项中都有这行数据，内存中的数据是最新的。

(3) 独占(Exclusive)。没有其他的 Cache 项包括这行数据，内存中的数据是最新的。

(4) 修改(Modified)。该项的数据是有效的，但内存中的数据是无效的，而且在其他 Cache 项中没有该项数据的拷贝。

下面我们结合图 13.13 来对 MESI Cache 一致性协议进行进一步的讨论，主要分析一下协议的工作过程。

当 CPU 刚刚启动的时候，所有的 Cache 项都标记为无效状态。第一次读取内存时，读入 CPU 的 Cache 的行的状态被标记为独占，因为此时它是 Cache 中的唯一的一份复制，如图 13.13(a)所示，CPU_1 读入了一行数据 A。接下来 CPU_1 的读操作都是从 Cache 中取得数据而不用经过总线了。其他的 CPU 也可以取相同的数据行并缓存，但是通过监听，最初的数据持有者 CPU_1 看到这行数据是自己已经有的就在总线上发布通告，宣布自己有一份该数据的复制。这样，这两个复制就都被标记成共享状态，如图 13.13(b)所示。CPU 对处于共享状态的 Cache 行的读操作不使用总线而且也不会产生状态改变。

如果 CPU_2 向共享状态的 Cache 行写入数据，它同时会把一个无效信号通过总线传送给其他的 CPU，通知它们把相应的数据复制置为无效。而 CPU_2 自己的 Cache 行的状态则变成了修改状态，如图 13.13(c)所示。此时，该行并不需要写回内存。这里需要注意的是，如果处于独占状态的 Cache 行发生了写操作，则不需要给其他的 Cache 发送无效信号，因为

其他 Cache 中并不存在该数据行的复制。

如果 CPU_3 读该行时,拥有该行的 CPU_2 知道内存中的数据是无效的,因此 CPU_2 就在总线上发送一个信号通知 CPU_3 等待它把该行写回内存。当写回操作完成后,CPU_3 从内存中取得数据的复制,然后把 CPU_2 和 CPU_3 的 Cache 中该行都标记为共享,如图 13.13(d)所示。在这之后,CPU_2 再次写该行,这将使 CPU_3 的 Cache 中对应行无效,如图 13.13(e)所示。

最后,CPU_1 再向该行中写入一个字。CPU_2 则将发生写操作,就在总线上发送一个信号通知 CPU_1 等待它把该行写回内存。当写回操作完成后,CPU_2 将自己 Cache 中的该行标记为无效,因此它知道 CPU_1 将会修改该行。这时发生的是 CPU_1 向没有缓存的行中写入数据的情况。如果使用了写分配策略,该行将被读入 CPU_1 的 Cache 并标记为修改状态,如图 13.13(f)所示。如果没有使用写分配策略,将直接对内存执行写操作,而且该行不会被读入到任何 Cache 中。

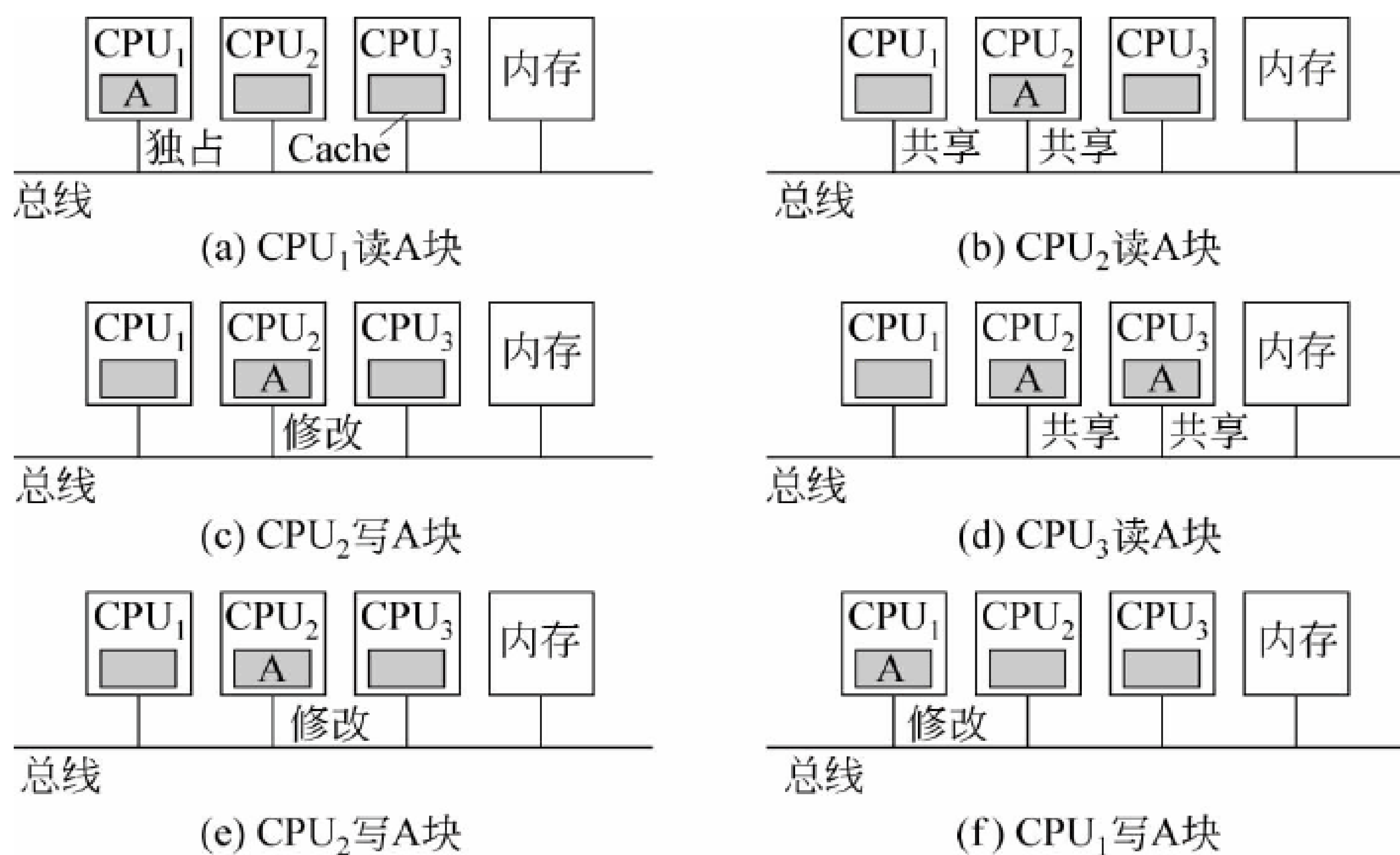


图 13.13　MESI 写回 Cache 一致性协议

3. 交叉开关和多级交换网络

由于总线带宽的限制,基于总线的 UMA 系统最多也就能使用 16 个或者 32 个 CPU。为了使用更多的 CPU,就需要采取其他形式的互联网络。比较常用的有两种方式,一种是使用交叉开关,另一种则是使用多级交换网络。下面分别对这两种方式进行简单介绍。

连接 n 个 CPU 和 k 个内存模块的最简单的电路就是交叉开关(Crossbar Switch),如图 13.14 所示。交叉开关技术在电话交换机中已经使用了数十年了,它可以按照任意的次序把输入线路和输出线路连接起来。

如图 13.14 所示,交叉开关中的每条水平线和垂直线的交点都是一个交叉点(Crosspoint)。一个交叉点就是一个小的交换结点,它的电路状态可以是打开或者关闭,具体状态取决于垂直线和水平线是否处于连接状态。在图 13.14 中可以看到有 3 个交叉点同时关闭,这就同时建立了 3 个 CPU 和内存的连接。

交叉开关网络是一种无阻塞的网络(Nonlocking Network),这是它最好的一个特性,这就意味着 CPU 不会因为某些交叉点或者线路被占用而无法与内存模块建立连接。而且,建立连接时不需要事先规划。对于图 13.14 中的交叉开关而言,即使已经建立了 7 个任意

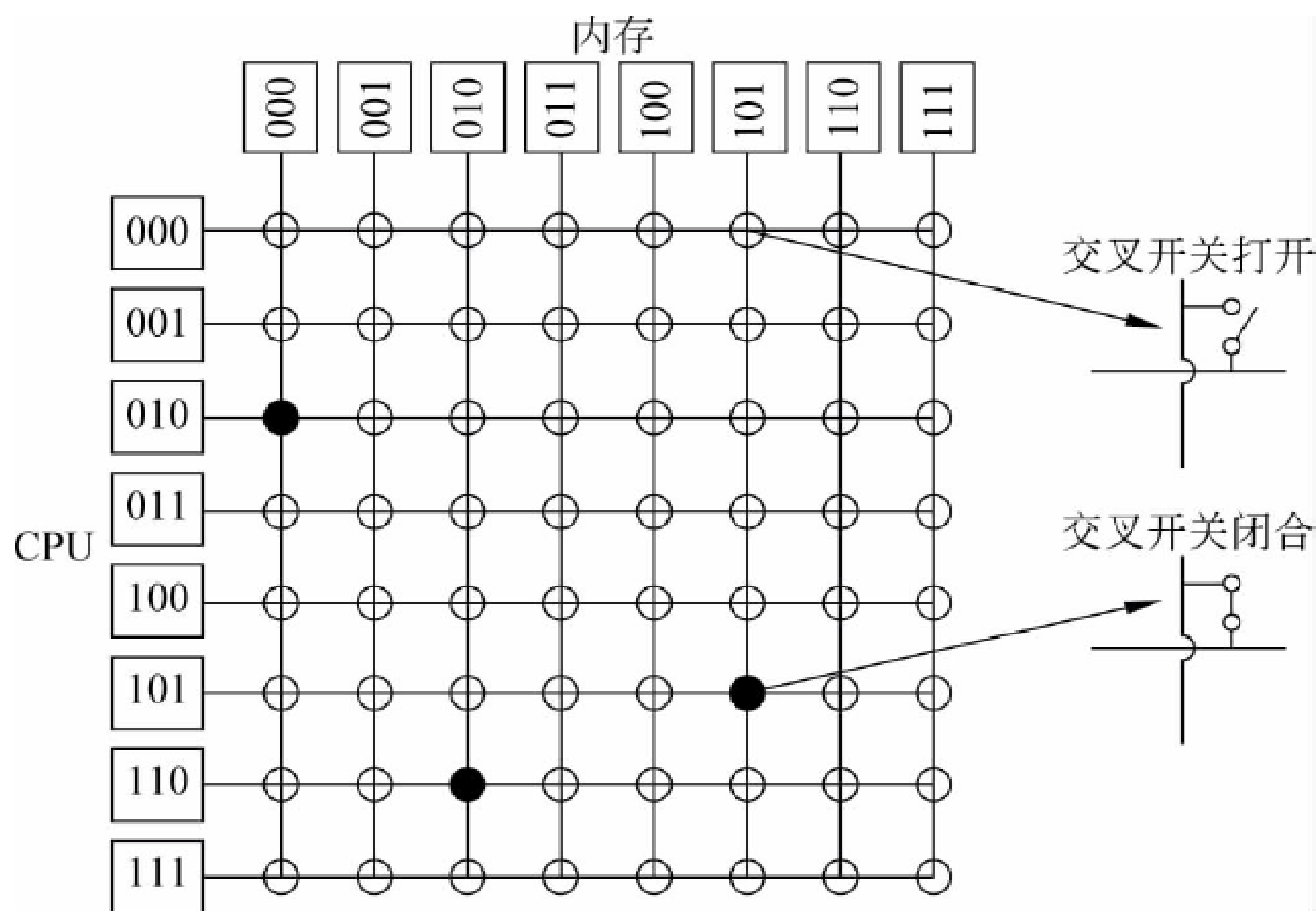


图 13.14　8×8 的交叉开关

的连接，仍然有可能在剩下的 CPU 和剩下的内存之间建立连接。

交叉开关也是存在缺点的，一个 $n\times n$ 的交叉开关有 n^2 个交叉点。如果需要使用 1000 个 CPU 和 1000 个内存模块，那么就需要 100 万个交叉点。这么多的交叉点是不可能实现的。当然，对于中等规模的系统，交叉开关设计是可行的。

如果想使用比 64 更多的 CPU，那么可能需要采取不同的策略。这个策略就是使用多级交换网络。我们首先介绍一个 2×2 的小交换结点，如图 13.15 所示。该交换结点具有两个输入和两个输出。从任意输入线到达的消息都可以交换到任意的输出线，消息可以包括以下 4 个部分。



图 13.15　2×2 的交换结点和消息格式

(1) 模块字段指出使用哪个内存模块。

(2) 地址定义了模块内的地址。

(3) 操作码指定操作，比如 READ 或者 WRITE。

(4) 可选的值字段可以包括一个操作数，例如 WRITE 操作要写入的 32 位字。

交换结点检查 Module 字段以判断消息应该通过 X 传递还是通过 Y 传递。

如图 13.16 所示使用了 12 个交换结点连接了 8 个 CPU 和 8 个内存模块。一般来说，如果有 n 个 CPU 和 n 个内存模块，就需要 $\log_2 n$ 级，每级 $n/2$ 个交换结点，一共是 $(n/2)\log_2 n$ 个交换结点，这要比前面所说采用交叉开关方式中需要 n^2 个交叉点的情况好多了，尤其当 n 比较大时更是如此。

可以采用 2×2 的交换结点组成比较大的多种不同形式的多级交换网络(Multistage Switching Networks)。Omega 网络就是其中比较经济实用的一种多级交换网络，Omega 网络采用被称为全混洗(Perfect Shuffle)的配线模式，即它的每级开关输入线与上一级开关输出线是均匀洗牌的连接方式。在这种连接方式中，就像洗扑克牌一样，全副牌分成两半，

再将两部分牌依序交错插入。如果用互连函数 $f(i)$ 表示这种连接关系的话，则 $f(i)=\{x_{n-2}x_{n-3}\cdots x_1x_0x_{n-1}\}$，即左端的二进制端号循环左移1位，就是连线右端的二进制端号。此外，还有其他多种洗牌方式。

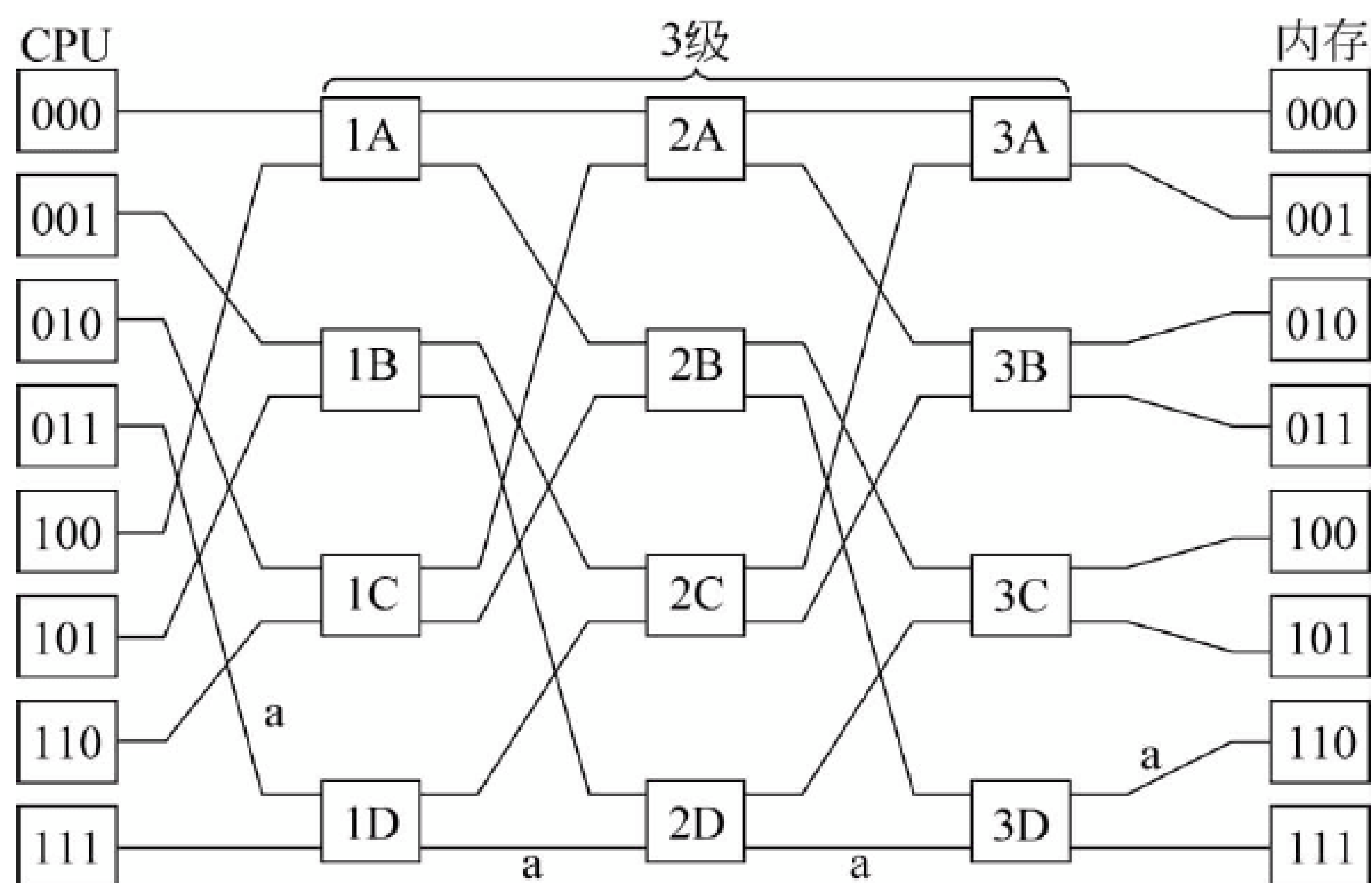


图 13.16　Omega 多级交换网络

为了理解 Omega 网络的工作原理，下面举一个例子进行说明。例如在图 13.16 所示的 Omega 多级交换网络中，CPU 011 需要从内存模块 110 中读取一个字。该 CPU 将发送一条模块字段为 110 的读消息给交换结点 1D。交换结点 1D 取出 110 最左面的位来确定如何传送这条消息。如果是 0 就从上面的输出线输出，如果是 1 就从下面的输出线输出。由于 110 最左面是 1，这条消息将传递给 2D。包括 2D 在内的所有第二级交换结点都使用第二位来确定如何发送消息。因为 110 的第二位还是 1，因此这条消息将发送给 3D。3D 继续使用第三位来决定消息发送的方向，第三位是 0。因此，该消息将从 3D 的上输出线输出到达内存模块 110。这条消息通过的路径在图 13.16 中用字母 a 表示。

如果两个内存访问请求使用的交换结点、链路和内存模块都不相同，那么它们是可以并行执行的。但是 Omega 网络是有阻塞的网络(Blocking Network)。在 Omega 网络中并非所有的请求都可以同步执行，当需要使用同一条链路或者同一个交换结点，或者请求访问同一个内存模块时都会发生冲突。

13.4.2　非一致性内存访问的 NUMA 多处理机系统

从上述的介绍可以知道，基于单总线的 UMA 多处理机系统的 CPU 数量最多也就几十个，而基于交叉开关或者多级交换网络的 UMA 多处理机系统需要大量昂贵的硬件，而且它们的 CPU 数量也不可能太多。为了构造超过 100 个 CPU 的多处理机系统，容易想到的一种思想就是放弃“所有的内存模块都有一致的访问时间”的限制，这一让步就产生了非一致性内存访问 NUMA 多处理机系统。

和 UMA 系统一样，NUMA 系统也为所有的 CPU 提供了单一的地址空间，但是和 UMA 不一样的是，在 NUMA 系统中访问本地内存模块比访问远程内存模块速度快。因此，所有为 UMA 计算机编写的程序都可以不加修改地在 NUMA 计算机上运行，但是在相同的时钟频率下，性能将低于 UMA 计算机。NUMA 计算机有 3 个区别于其他的多处理机系统的关键的特点。

(1) 所有的 CPU 都看到一个单一的地址空间。

(2) 使用 LOAD 和 STORE 指令访问远程内存。

(3) 访问远程内存比访问本地内存慢。

没有使用 Cache 的 NUMA 系统被称为 NC-NUMA,也就是说这种系统中不隐藏远程内存的访问时间。如果使用了 Cache,那么系统就被称为 CC-NUMA。

图 13.17 所示的是一种基于两级总线的 NUMA 计算机。它由一组 CPU 组成,每块 CPU 都有可以通过局部总线访问的内存模块,各个结点之间通过系统总线相连。当一个内存访问请求进入内存管理单元 MMU 之后,MMU 将检查请求访问的字是否在本地内存中。如果是,请求将通过局部总线发送到本地内存去读取该字。如果不是,请求将通过系统总线发往拥有该字的模块,然后由拥有该字的模块发回响应信息。显然,后者所花费的时间肯定比前者长。当一个程序在远程内存上运行时,它将比在本地内存上运行的相同的程序多花 10 倍的时间。

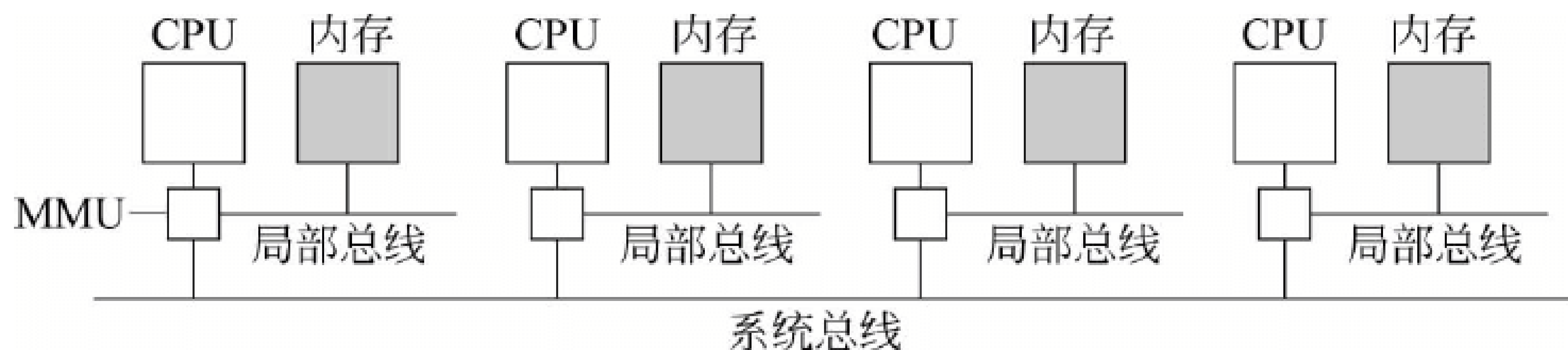


图 13.17　基于两级总线的 NUMA 计算机

在 NC-NUMA 系统中,因为没有使用 Cache,所以内存的一致性是有保证的。每个内存字都只在一个地点存在,因此不存在包含陈旧数据的复制的问题。然而,哪个页面位于哪个内存模块中是很重要的,如果内存页面在错误的位置上,那么性能的损失将相当大。因此,NC-NUMA 使用了精心设计的软件来移动页面以使性能达到最佳。

图 13.17 所示的多处理机系统因为它们没有使用 Cache,所以这样的系统可扩展性很差。每次使用非本地内存中的字都要去访问远程内存将严重地影响性能。但是,如果使用了 Cache,就必须保证 Cache 的一致性。保证 Cache 一致性的一种办法是监听系统总线,但是这种方法不适合于建造大型的多处理机系统,例如在多级网络上实现广播功能的代价很大,所以监听协议就无法使用。把使其他数据块无效的一致性命令只发给存放相应数据块的 Cache 是一个很好的解决办法,这就产生了一种完全不同的新策略,即基于目录(Directory Based)的 Cache 一致性协议。图 13.18 给出了一个具有 256 个结点的基于目录的多处理机系统示意图。

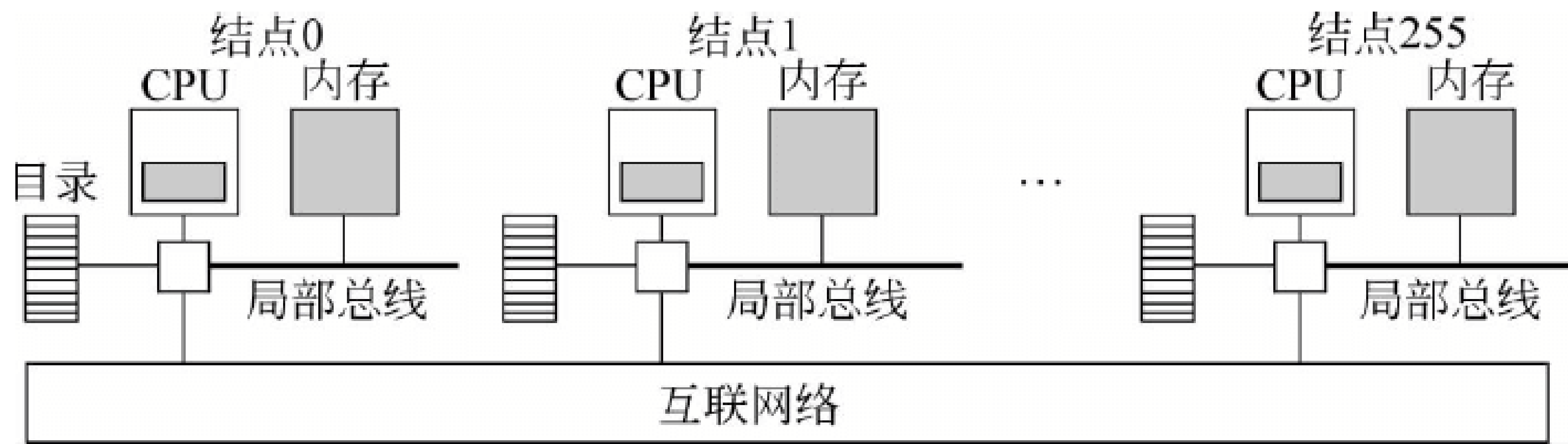


图 13.18　256 个结点的基于目录的多处理机系统

目前建造大型的 CC-NUMA 多处理机系统最常用的策略是基于目录的多处理机系统。

其主要设计思想是维护一个数据库记录内存每个Cache行在什么位置上以及状态是什么。当某个Cache行被访问时,将查询该目录数据库找到该Cache行的位置以及该行是干净的还是脏的(也就是被修改过的)。由于每条访问内存的指令都要查询这个数据库,因此这个数据库必须保存在速度非常快的特定硬件中以保证能够在不到一个总线周期的时间内做出响应。

13.4.3 基于Cache内存访问的COMA多处理机系统

NUMA和CC-NUMA计算机有相同的缺点,它们访问远程的内存比访问本地内存速度慢。在CC-NUMA计算机中,通过使用Cache在某种程度上隐藏了这种性能的差别。无论怎样,如果需要使用的远程数据超过了Cache的能力,Cache不命中的情况将会频繁出现,性能也会急剧下降。

因此我们面对的情况是UMA计算机性能很好但是规模受到一定的限制,而且价格昂贵。NC-NUMA计算机可扩展性较好,但是需要手动或者半自动地放置页面,通常使用混合型。问题在于很难预测需要使用的页面的位置,而且在任何一种情况下,页面作为一个移动的单元来说都太大了。CC-NUMA计算机,例如Sun Fire E25K,当许多CPU都需要大量的远程数据时,性能将会变得很差。总而言之,每种设计方案都有很大的局限性。

另一种类型的多处理器系统试图通过把每个CPU的主存作为Cache来解决这些问题。在这种称为COMA的多处理器系统中,页面并不像NUMA和CC-NUMA计算机中的页面那样固定在宿主计算机中。

在COMA系统中,物理地址空间被划分成Cache块,这些块根据需要在系统中来回移动。Cache块不再有宿主计算机了。只存放需要的块的内存称为吸引内存。把主存作为一个大的Cache可以极大地提高命中率,从而提高性能。

COMA系统存在两个新的问题,第一个问题是如何对Cache块进行定位;第二个问题是当需要把某块从内存中清除掉时,如果该块是最后一个复制如何处理。要想使COMA有效地工作,必须要解决这两个问题。目前已经有多种不同的方案,读者可以阅读一些相关的参考资料。

13.5 基于消息传递的多计算机系统

通过前面介绍的多处理机系统可以看到,从操作系统角度来看,多处理机系统提供了能够使用通常的LOAD和STORE指令存取的共享内存。而且这种共享内存可以用多种方式实现,包括监听总线、数据交叉开关、多级交换网络和各种基于目录的机制。无论采用何种机制,为多处理机系统编写的程序可以访问内存的任何位置而不用知道内存的内部拓扑结构和实现机制。这也是多处理机系统的优点所在。

然而多处理机系统也有它自身的限制。首先而且最重要的一点是多处理机系统很难扩展到很大的规模。例如Sun使用了大量的硬件才使Enterprise 10000扩展到了64个CPU。Sequent NUMA-Q可以扩展到356个CPU,但是它是以不一致的内存访问时间为代价的。另外,多处理机系统中的内存争用对性能的影响很大。如果100个CPU经常读写同一个变量,那么对内存模块、总线和目录的争用将使性能受到很大的影响。

我们在本章开始的时候已经提到了 MIMD 并行处理器系统的另一类，即多计算机系统。这类系统能够使用 2048 甚至 9416 个 CPU。也许经过若干年的努力，人们或许可以造出 10000 个结点的商用多处理机系统，但是到那时候，人们已经可以使用 100000 结点的多计算机系统了。

在多计算机体系结构中，每个 CPU 都有自己的私有内存，私有内存只能供自己使用而其他的 CPU 则不能访问。这种体系结构有时也被称为分布式内存系统（Distributed Memory System），如图 13.19(a)所示。也就是说，多处理机系统所有的 CPU 共享一个单一的物理地址空间，而在多计算机系统中，每个 CPU 都有自己独立的物理地址空间。

由于多计算机系统中的 CPU 不能通过读写共享内存进行通信，它们需要另一种不同的通信机制。在多计算机系统中，通信是通过使用互联网络传递消息来实现的。多计算机的例子包括 IBM BlueGene/L、Red Storm 和 Google 集群等。

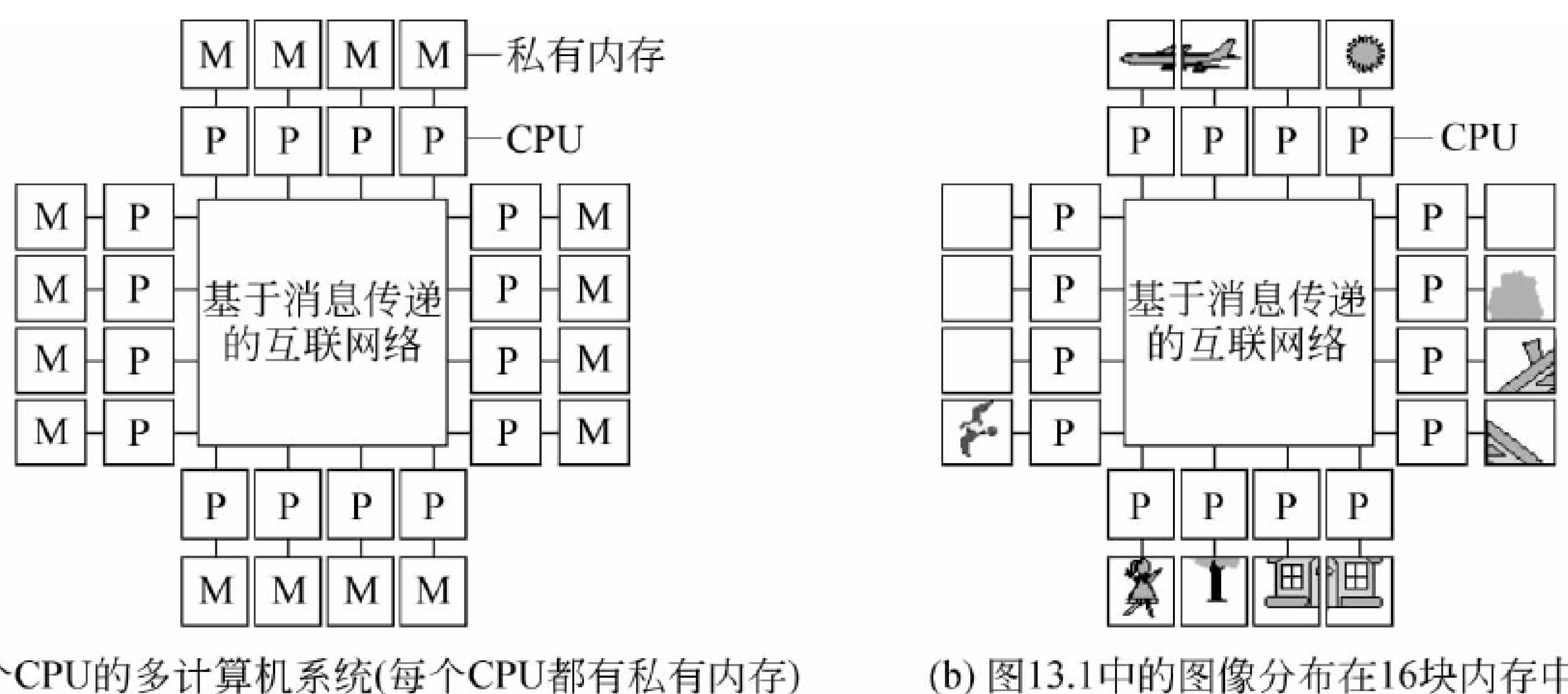


图 13.19　多计算机设计方案

多计算机系统中没有硬件实现的共享内存这一特点也在很大程度上影响了其软件体系结构。多处理机系统中多个处理器共享一个单一的地址空间，所有的处理器都可以通过执行 LOAD 和 STORE 指令来访问所有的内存，而这一点在多计算机系统中是不可能做到的。举例来说，如果图 13.11(b)中的 CPU_0（最上面一排左手第一个 CPU）发现它分析的图像中的对象扩展到了分配给 CPU_1 的图像中，它就可以继续读内存来访问 CPU_1。然而如果图 13.19(b)中的 CPU_0 也有同样的发现，它就不能直接读 CPU_1 的内存，在如何获取需要的数据方面，这两种体系结构之间存在着很大的不同。

在多计算机系统中，如果一个 CPU 发现某个其他的 CPU 那里有它需要的数据，它就给该 CPU 发送一条请求获得数据的消息。一般来说，发请求消息的 CPU 将阻塞（也就是等待）直到请求被响应。在上面的例子中，当消息到达 CPU_1 后，CPU_1 的软件将分析该消息并把需要的数据发送回来。当响应消息到达 CPU_0 后，软件将解除阻塞并继续执行。

在多计算机系统中，进程间通信通常使用 Send 和 Receive 这样的软件原语。因此，多计算机系统中的软件结构就和多处理机系统不同，而且比多处理机系统复杂得多，这同时也就导致了多计算机系统的编程比多处理机系统的编程要复杂得多。

那么，为什么不都采用容易编程的多处理机系统，还要去设计多计算机系统呢？答案很简单，就相同数量的 CPU 来说，大规模的多计算机系统比多处理机系统结构简单而且造价便宜。实现一台具有数百个 CPU 的共享内存的计算机是一项很复杂的工作，而建造一台

具有10000个或者更多的CPU的多计算机系统则相对而言是一项比较简单的工作。

多计算机系统中的每个结点都由一个或者多个CPU、RAM、磁盘以及其他的输入/输出设备和通信处理器组成。通信处理器通过互联网络相互连接起来。可以使用多种不同的拓扑结构、交换策略和寻径算法。所有的多计算机系统的一个共同的特点是当应用程序执行Send原语时,系统就会通知通信处理器把一块用户数据传递到目的计算机中。图13.20是多计算机系统的通用结构。

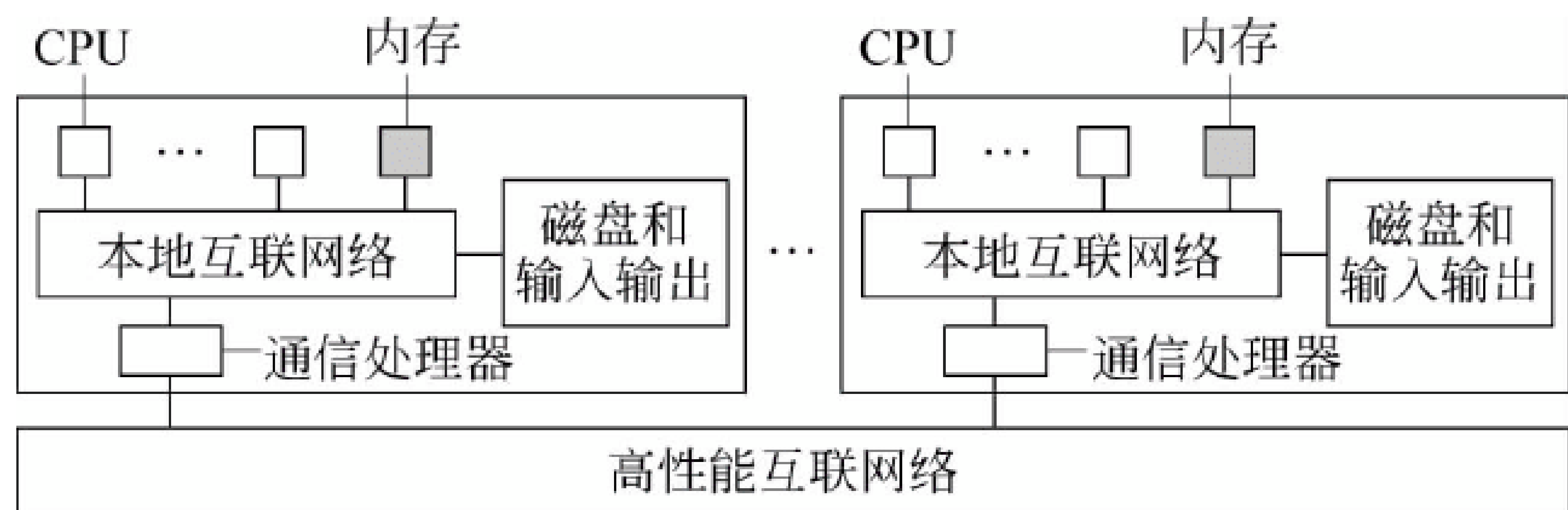


图13.20 通用多计算机体系结构

多计算机系统有各种不同的规模,因此对它们进行清晰的分类是比较困难的。但是一般来说,可以把多计算机系统分成两大类:MPP和COW。下面我们来依次介绍这两种系统。

13.5.1 大规模并行处理机

第一类多计算机系统是大规模并行处理系统(Massively Parallel Processors,MPP),它们被用于科学计算、工程计算和其他需要大量计算的工业部门,每秒可以处理大量的事务,还可以用于数据仓库等,这是一种价值数百万美元的超级计算机系统。最初,MPP主要用于科学计算领域,现在它们大多数都用于商务环境。在很大程度上说,MPP系统取代了SIMD计算机(向量超级计算机和阵列处理机)原来的位置。

大多数的MPP系统都使用标准的商用CPU作为它们的处理器。比较常用的有Intel公司的Pentium系列、Sun公司的UltraSPARC和IBM公司的PowerPC。一般来说,MPP系统具有以下几个特点。

首先,MPP系统使用了高性能的私用的互联网络,可以在低延时和高带宽的条件下传递消息,这很重要,因为大部分信息都很小(通常小于256个字节),但是大部分的流量是来自于比较大的消息(超过8KB)。MPP系统还需要使用大量定制的软件和库。

其次,MPP系统具有强大的输入/输出能力。需要使用MPP解决的问题往往要处理大量的数据,常常会达到太字节。这些数据必须分布在多个磁盘上而且需要在结点之间以很高的速率传递。

最后,MPP系统能够进行特殊的容错处理。在使用数千个CPU的情况下,每星期有若干个CPU失效是不可避免的。但是如果因为一个CPU崩溃导致一个运行了18个小时的任务被取消是无法接受的,尤其是当这种情况每周都会出现时。因此大规模的MPP系统总是使用特殊的硬件和软件来监控系统,监测错误并从错误中平滑地恢复。

MPP的一般设计原理比较简单而且容易理解,其实MPP就是由快速的互联网络连接起来的比较标准的计算结点的组合。

13.5.2 工作站集群

工作站集群(Clusters Of Workstations,COW)是另一种多计算机系统,它也被称为工作站网络(Network Of Workstations,NOW)。一般来说,COW系统是由数百台PC或者工作站通过商用网络连接在一起构成的。

从体系结构上讲,COW有两点和MPP不同。第一,COW的结点是更完整的计算机,计算机可以是同构的也可以是异构的。结点都有自己的磁盘,驻留有自己的操作系统;并且,一般都有一定的自主性。结点计算机脱离COW照样能运行。第二,MPP使用制造厂商专有(或者说是有专利权)的高速通信网络;COW一般采用公开销售的标准高速局域网或系统域网,网络通常是与结点计算机的I/O总线相连。

技术是COW发展的推动力量。MPP系统中使用的CPU就是商用的CPU,这些商用的CPU包括Alpha系列、Pentium系列等,任何人都可以买到。它们使用的DRAM也很常用,而且运行的是UNIX操作系统。总而言之,在这些方面COW和MPP没有什么区别。MPP中的特殊部分在于它的高速互联网络,但是现在商用的高速互联网络也已经开始出现。COW和MPP相比,优势在于COW可以完全使用可以买到的商用组件装配而成,这些商用组件都是大规模生产的产品,因此能够获得较高的性价比。而且由于存在市场竞争,这些组件的性价比还在不断提高,所以COW有可能取代MPP系统的位置。

COW系统有许多种,其中占主导地位的主要有两种:集中式的和分散式的。集中式的COW是装在一个大机架上的工作站或者PC的集群。有时候它们排列的很紧密以节省物理空间和光纤长度。一般来说,这些计算机的种类都是相同的,而且除了网卡和磁盘之外没有其他的外设。分散式的COW是由分布在一座大楼或者校园里的工作站或者PC组成的,通常它们是通过局域网连接的。一般来说,这些计算机的种类是不同的,而且有丰富的外设。最重要的是,这些计算机属于不同的所有者,只能在这些计算机空闲的时候并且所有者同意的情况下其他用户才能使用它们。使用空闲的工作站组成COW意味着当计算机的所有者要求回收他们的计算机时,要把正在运行的任务在不同的计算机之间转移。任务转移是可以实现的但是会增加软件的复杂性。

本章内容小结和学习方法建议

并行计算机系统可以分成两大类:SIMD系统和MIMD系统。SIMD计算机可以同时在多个数据集上并行执行同一条指令。这种类型的计算机包括阵列处理器系统和向量处理机。MIMD计算机在并行计算领域占有统治地位,它可以同时在不同的结点计算机上运行不同的程序。MIMD计算机可以分成多处理机系统和多计算机系统两大类,多处理机系统共享物理内存,而多计算机系统中物理内存是不被共享的。无论是多处理机系统还是多计算机系统,都通过不同的高速互联网络把一大批CPU和内存模块连接起来。

设计并行计算机的目的就是使它的处理能力比单处理器的计算机更强。并行计算机体系结构的性能问题主要涉及硬件和软件性能指标,以及如何获得更高的性能等方面。软件问题在讨论并行计算机体系结构的时候也是非常重要的,如果没有并行软件,并行硬件基本上就没有什么用处。

因为在大多数多处理机系统中内存都被分成了多个不同的模块，所以根据共享内存的模块组织方式，可以把多处理机系统分成3类，分别是一致性内存访问计算机(UMA)、非一致性内存访问计算机(NUMA)和基于Cache内存访问计算机(COMA)。而多计算机系统可以粗略地分成MPP系统和COW系统，它们的界限有时也很模糊。MPP是高度商业化的系统，它们都使用专用的高速互联网络，COW则是建立在商用组件基础上的。

应该说这一章所提供的是计算机系统结构课程的核心内容，学起来会感到有点难度，强调建立起并行计算机系统的基础概念，通过构建大型、超大型的计算机系统提高计算机处理能力的多种可行方案，还必须从硬件、软件两个方面来解决问题。

习题与思考题

1. 简述计算体系结构的两个发展方向。说明计算体系结构的Flynn分类法的分类依据，这种分类中各类计算机的结构特点是什么?

2. 为什么并行计算机系统能够得到快速的发展?

3. 互联网络的交换结点一般有哪几种设计方案? 解释各种方案的工作过程并说明各种不同方案的优点和缺点。

4. 有一个8×8×8的立方体互联网络，每条链路的全双工带宽是1GB/s。请问网络的对分带宽是多大?

5. Amdahl定律限制了并行计算机能够获得的最大的加速比。我们把函数f定义为在CPU数量不限的情况下程序能达到的最大的加速比。请问$f=0.1$是什么含义?

6. 图13.8是总线不能扩展而网格网络成功扩展的例子。假定每条总线或者链路的带宽为b，请计算图中4种情况下每个CPU的平均带宽。然后把系统扩展到64个CPU并重复上面的计算过程。当CPU的数量趋于无穷时，系统将受到什么限制?

7. 假定有n个CPU连接在同一条总线上。在某个给定的时间段里，任何一个CPU使用总线的概率都是p。计算出现下列情况的出现概率：①总线空闲(没有任何请求)。②只有一个请求。③多于一个请求。

8. 在MESI Cache一致性协议中存在某个Cache行确实位于本地Cache中但是却需要通过总线来寻找该行的情况吗? 如果存在这种情况，请解释其原因。

9. MESI Cache一致性协议有4种状态，其他的写回式Cache一致性协议都只有3种状态。MESI中的哪个状态可以不要呢? 这样做的结果是什么? 如果你只能选择3种状态，你会选择哪3种?

10. 一个Omega网络连接了4096个RISC CPU和4096个访问速度无穷快的内存模块，每个CPU的周期是60ns。每个交换部件的延迟时间是5ns。那么一条LOAD指令需要多少个延时槽?

11. MPP系统与COW系统有哪些相同点和不同点?

参 考 文 献

[1] 王诚，刘卫东，宋佳兴．计算机组成与设计[M]．第3版．北京：清华大学出版社，2008.

[2] Andrew S. Tanenbaum 著．刘卫东，宋佳兴译．计算机组成结构化方法[M]．第6版．北京：人民邮电出版社，2014.

[3] David A. Patterson，John L. Hennessy 著．王党辉等译．计算机组成与设计：硬件/软件接口[M]．第5版．北京：机械工业出版社，2015.

[4] 刘卫东．计算机组成与结构[M]．北京：机械工业出版社，2003.

[5] 张晨曦，王志英等．计算机体系结构[M]．北京：高等教育出版社，2000.

[6] 胡越明．计算机组成与设计[M]．第3版．北京：科学出版社，2006.

[7] 郑伟民，汤志忠．计算机系统结构[M]．第2版．北京：清华大学出版社，1998.

[8] 袁春风．计算机组成与系统结构[M]．第2版．北京：清华大学出版社，2015.

[9] 王诚，宋佳兴，张改革，李山山．计算机组成与体系结构（第3版）实验教程．北京：清华大学出版社，2017.